Integer Programming and Combinatorial Optimization

Proceedings of a Conference held at the University of Waterloo, May 28-30 1990, by the Mathematical Programming Society

edited by Ravi Kannan and W.R. Pulleyblank

Integer Programming and Combinatorial Optimization

Proceedings of a Conference held at the University of Waterloo,
May 28-30 1990, by the Mathematical Programming Society

Ravi Kannan
Department of Computer Science
Carnegie-Mellon University

W R Pulleyblank
Department of Combinatorics and Optimization
University of Waterloo

University of Waterloo Press

ISBN 0-88898-099-X

University of Waterloo Press
Porter Library
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
Phone: 519 885-1211, ext. 3369
FAX: 519-747-4606

Printed and bound at Graphic Services, University of Waterloo

Canadian Cataloguing in Publication Data

Main entry under title:

Integer programming and combinatorial optimization

Includes bibliographical references.
ISBN 0-88898-099-X

1. Integer programming – Congresses. 2. Combinatorial optimization – Congresses. I. Kannan, Ravi, 1953-
II. Pulleyblank, W. R. III. Mathematical Programming Society (U.S.)

QA402.5.159 1990 519.3 C90-094335-1

CONTENTS

Preface

The idea of organizing a conference on integer programming and combinatorial optimization which would consist of papers selected by a program committee and presented in a single stream had its origin in 1987. At that time we were both visiting the Institut für Operations Research at the University of Bonn. We recognized the role played in the computer science community by the FOCS and STOC meetings, organized by the IEEE and ACM, respectively. These meetings provide a forum for the presentation of results selected by a committee of senior researchers on the basis of extended abstracts. We thought that it would be interesting to hold a meeting with a similar format in our scientific community.

We discussed the idea with the Executive Committee of the Mathematical Programming Society and received strong encouragement. The Society offered to be a sponsor of the meeting and agreed to organize it as an official society meeting. The Department of Combinatorics and Optimization of the University of Waterloo agreed to be the host for the meeting.

In 1988 we formed the program committee, whose membership is listed below, and the meeting was formally announced at the Mathematical Programming Society in Tokyo, Japan, at the end of August 1988. We are very grateful to the members of the program committee for agreeing to serve, and for the time and effort spent on the selection of the papers for presentation.

The deadline for submission of extended abstracts was November 15, 1989. We were delighted by the response of the community. We received 76 submissions, of which the majority were of a very high quality. We were also very pleased at the mixture of papers from senior members of our community as well as young researchers. Abstracts came from North and South America, Europe, and Asia.

All abstracts were sent to all program committee members. Four members of the program committee were able to meet in early January in West Germany. The other members sent in their recommendations, and in mid January we met at Carnegie Mellon University to make the final selections. This was, not surprisingly, very difficult. Because of our decision to have a single string of presentations, we were limited to accepting 36 papers. This forced us to reject papers which we feel are of publishable quality. The main criteria for acceptance were the perceived interest of the results plus the originality of the material.

This volume contains the full text of all but three of the accepted papers; for these we include the extended abstracts. In addition, in one case, the author (W. Hochstättler) of an accepted paper was forced to cancel his participation, but requested that we include his paper. We agreed to this.

These papers contained in this volume have not been refereed. The papers were provided in camera ready form by the authors. It is our expectation that revised versions of most will appear in refereed journals.

We gratefully acknowledge financial support from the Mathematical Programming Society, the Information and Technology Research Centre of the Province of Ontario, The Natural Sciences and Engineering Research Council of Canada, Canadian Pacific Ltd. and the University of Waterloo. We also wish to express our sincere thanks to Tracy Taves, Kim Gingerich and Marg Feeney of the Department of Combinatorics and Optimization of the University of Waterloo for all their help in organizing the meeting and preparing these proceedings.

Ravi Kannan
Pittsburgh, Pennsylvania, U.S.A.

William Pulleyblank
Waterloo, Ontario, Canada

Program Committee
V. Chvátal, Rutgers University
W. H. Cunningham, Carleton University
R. Kannan, Carnegie-Mellon University
R.M. Karp, University of California, Berkeley
G.L. Nemhauser, Georgia Institute of Technology
W.R. Pulleyblank, University of Waterloo
P.D. Seymour, Bell Communications Research

Integer Programming and Combinatorial Optimization Conference
May 28 - 30, 1990
Conference Program

MONDAY A.M.(1) **(9:00-10:30 a.m.)**

1. E.C. Sewell & L.E. Trotter, Jr. - *On Rank Facets for the Stable Set Polytope*
2. M. Conforti & G. Cornuejols - *A Decomposition Theorem for Balanced Matrices*
3. A. Frank - *Conservative Weightings and Ear-Decompositions of Graphs*

MONDAY A.M. (2) **(11:00 a.m.-12:30 p.m.)**

1. S. Onn - *On the Radon Number of the Integer Lattice*
2. A. Sebö - *Hilbert Bases, Caratheodory's Theorem and Combinatorial Optimization*
3. A.A. Fridman & E.V. Levner - *A Survey of Some Results in Discrete Optimization*

MONDAY P.M.(1) **(2:00-3:30 p.m.)**

1. L. Hall & D. Shmoys - *Near-Optimal Sequencing with Precedence Constraints*
2. J.A. Hoogeveen - *Minimizing Maximum Earliness and Maximum Lateness on a Single Machine*
3. S.L. van de Velde - *Dual Decomposition of Single Machine Scheduling Problems*

MONDAY P.M. (2) **(4:00-5:30 p.m.)**

1. C.K. Cheng & T.C. Hu - *Ancestor Tree for Arbitrary Multi-Terminal Cut Functions*
2. S. Sridhar & R. Chandrasekaran - *Integer Solution to Synthesis of Communication Networks*
3. D.S. Hochbaum - *Optimal Algorithms for the Allocation Problem and Its Extensions*

TUESDAY A.M. (1) **(9:00-10:30 a.m.)**

1. D. Bienstock - *Some Provably Hard Crossing Number Problems*

2. V.V. Vazirani - *A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ General Graph Matching Algorithm*

3. F. Aurenhammer, J. Hagauer & W. Imrich - *Finding the Prime Factors of a Cartesian Product Graph G in $O(|E(G)| \log |V(G)|)$ Time*

TUESDAY A.M. (2) **(10:30-11:00 a.m.)**

1. B. Jaumard, P. Hansen & M. Poggi de Aragaö - *Column Generation Methods for Probabilistic Logic*

2. M.A. Trick - *Scheduling Variable-Speed Machines*

3. V.L. Beresnev & A.A. Ageev - *Polynomially Solvable Cases of the Simple Plant Location Problem*

TUESDAY P.M. (1) **(2:00-3:30 p.m.)**

1. A. Boyd - *New Polyhedral results for the precedence constrained knapsack problem*

2. N.Ascheuer, L.F. Escudero, M. Groetschel & M. Stoer - *On Identifying in Polynominal Time Violated Subtour Elimination and Procedence Forcing Constraints for the Sequential Ordering Problem*

3. S. Chopra & G. Rinaldi - *The Graphical Asymmetric Traveling Salesman Polyhedron*

TUESDAY P.M. (2) **(4:00-5:30 p.m.)**

1. B. Reed - *Perfection, Parity, Planarity, and Packing Paths*

2. W-L Hsu - *Efficient Isomorphism Algorithms for Circle Graphs and Circular–Arc Graphs*

3. A. Recski - *Vertex Disjoint Channel Routing on Two Layers*

WEDNESDAY A.M. (1) **(9:00-10:30 a.m.)**

1. M. Dyer & A. Frieze - *Probabilistic Analysis of the Generalized Assignment Problem*

2. M.H. Schneider - *Max-Balanced Flows*

3. P.M. Camerini, G. Galbiati & F. Maffioli - *Modular Arithmetic and Randomization for Exact Matroid Problems*

WEDNESDAY A.M. (2) **(11:00 a.m.-12:30 p.m.)**

1. S. Hadley, F. Rendl & H. Wolkowicz - *Bounds for the Quadratic Assignment Problem*

2. S. Khuller & J. Naor - *Flow in Planar Graphs with Vertex Capacities*

3. E.A. Dinic - *The Fastest Algorithm for the Pert Problem with AND-and OR-Nodes (The New-Product-New Technology Problem)*

WEDNESDAY P.M. (1) **(2:00-3:30 p.m.)**

1. E. Balas - *On Determining Whether a Valid Inequality is Facet Defining*

2. J. Fonlupt & A. Sebö - *On the Clique-rank and the Coloration of Perfect Graphs*

3. M. Deza & M. Laurent - *New Results on Facets of the Cut Cone*

WEDNESDAY P.M. (2) **(4:00-5:30 p.m.)**

1. E.J. Anderson & M.C. Ferris - *A Genetic Algorithm for the Assembly Line Balancing Problem*

2. N. Karmarkar - *An Interior-Point Approach to NP-complete Problem*

3. A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan & M.G.C. Resende - *Computational Experience with an Interior Point Algorithm on the Satisfiability Problem*

POLYNOMIALLY SOLVABLE CASES OF THE SIMPLE PLANT LOCATION PROBLEM

Alexander A. AGEEV and Vladimir L. BERESNEV
Institute of Mathematics, Novosibirsk, USSR

ABSTRACT. The paper deals with the simple (uncapacitated) plant location problem that is well-known to be *NP*-hard. Non-trivial polynomially solvable cases of the problem are thus of particular interest. We give a brief survey of some investigations in this direction which have been published in Russian and may be widely unknown in the West.

The simple plant location problem (SPLP) is usually formulated as follows:

$$\text{minimize} \sum_{i\in I} c_i y_i + \sum_{i\in I}\sum_{j\in J} d_{ij} x_{ij} ,$$

$$\text{subject to} \sum_{i\in I} x_{ij} = 1 , \quad j\in J,$$

$$0 \le x_{ij} \le y_i , \quad i\in I, \ j\in J,$$

$$x_{ij}, y_i \in \langle 0,1\rangle , \quad i\in I, \ j\in J,$$

where $I=\langle 1,\ldots,m\rangle$ is a set of possible locations of suppliers, $J=\langle 1,\ldots,n\rangle$ is a set of consumers, $c_i \ge 0$ is a fixed cost of establishing a supplier on the site i, $d_{ij} \ge 0$ is a cost of transporting the demanded amount of some product from the supplier on the site i to the consumer j. Provided that all the demands must be satisfied one needs to locate suppliers on the sites so as to minimize the total cost.

The SPLP is well-known to be *NP*-hard. Non-trivial polynomially solvable cases of the problem are thus of particular interest. We give a brief survey of some

investigations in this direction which have been published in Russian and may be widely unknown in the West.

There are several ways to produce special cases of the SPLP. One of them is to impose restrictions on the class of matrices (d_{ij}).

A matrix (d_{ij}) is said to be *p-connected* $(p=1,2,\ldots)$ if for each pair $i_1, i_2 \in I$ the sequence $(d_{i_1 j} - d_{i_2 j})$ $(j=1,\ldots,n)$ has at most p sign alterations.

The class of 1-connected (originally named connected) matrices was first discovered and explored by Gimadi in [Gi69] (see also [BGD78]). He presented an $O(mn^2)$ algorithm for the SPLP with these matrices. The algorithm is based on a polynomial-time reduction of this special case to the well-solved nearest neighbor problem [BD62].

The concept of p-connectedness was introduced by Beresnev in [Be79]. He suggested a linear algorithm for the former case and an $O(mn^3)$ algorithm for the SPLP with 2-connected matrix (d_{ij}). Both of them are of dynamic programming type. Later Ageev [Ag83] proved that the SPLP with 3-connected matrix is *NP*-hard.

In [Gi83] Gimadi introduced (for the case of trees) the following concept which is closely related to that of p-connectedness.

Let $G=(J,E)$ be a simple connected undirected graph with vertex set J and edge set E.

A matrix $(d_{ij})(i \in I, j \in J)$ is said to be *connected with respect to the graph* G if for each pair $i_1, i_2 \in I$ there exists a partition (J_1, J_2) such that the subgraphs induced by J_1 and J_2 are connected and in addition

$$d_{i_1 j} \le d_{i_2 j} \quad \text{for all } j \in J_1,$$

$$d_{i_2 j} \le d_{i_1 j} \quad \text{for all } j \in J_2.$$

Consider the SPLP with the input $\langle m, n,$ a vector (c_i), a connected undirected graph $G = (J, E)$, a matrix (d_{ij}) connected with respect to $G \rangle$. Note that any class of connected graphs $\mathcal{G}$ induces a certain special case of the SPLP (we call it the SPLP on $\mathcal{G}$). It is easy to observe that if a matrix (d_{ij}) is 1-connected (2-connected) then it is connected with respect to the chain $(1,\ldots,n)$ (the cycle $(1,\ldots,n,1)$). It follows from the results mentioned above that the SPLP's on chains and on cycles are polynomially solvable.

The main contribution made by Gimadi in [Gi83] was an $O(mn)$ algorithm for the SPLP on trees. Let us pause at this point to indicate two consequences of this result.

The first one arises from the fact that the SPLP on a class of graphs is an extension of optimization version of the well-known k-median problem on this class of graphs. The latter can be formulated as follows. Given an undirected graph G with nonnegative vertex weights $c(j)$ and edge lengths $l(e)$, one needs to find a non-empty subset $M \subseteq J$ minimizing

$$\sum_{j \in M} c(j) + \sum_{j \in J \setminus M} \min_{k \in M} d(j,k),$$

where $d(j,k)$ is the length of the shortest path connecting the vertices j and k. For trees this problem can be solved in $O(n^3)$ time by the Trubin's algorithm [Tr76]. Later this algorithm was rediscovered by Kolen [Ko83]. The Gimadi's algorithm solves this special case in $O(n^2)$ time.

Another consequence is related to the set covering problem. It is easy to observe that the weighted set covering problem

$$\min \sum_{i\in I} c_i y_i$$

$$\text{s.t.} \quad \sum_{i\in I} a_{ij} y_i \geq 1, \quad j\in J,$$

$$y_i \in \{0,1\}, \quad i\in I,$$

where $c_i \geq 0$, $a_{ij}\in\{0,1\}$, $i\in I$, $j\in J$, is equivalent to the special case of the SPLP in which

$$d_{ij} = \begin{cases} 0 & \text{if } a_{ij}=1, \\ \sum_{i\in I} c_i + 1 & \text{if } a_{ij}=0. \end{cases}$$

If the constraint matrix (a_{ij}) is connected with respect to a graph $G=(J,E)$ then so is the matrix (d_{ij}). Hence the set covering problem for the class of matrices connected with respect to trees can be solved by the Gimadi's algorithm in $O(mn)$ time. It is easy to check that this class of binary matrices contains the class of the fork-free ones introduced in [BEW84].

The latest results on this subject are due to Ageev [Ag89], who has designed a polynomial-time transformation from the SPLP on outerplanar graphs to the SPLP on cycles and, as a consequence, an $O(mn^3)$ algorithm for the former problem. This transformation is based on the theorem asserting that the class of matrices connected with respect to outerplanar graphs coincides with the class of matrices connected with respect to cycles. It follows that the latter contains the class of matrices connected with respect to trees. It also follows that the k-median problem (and its optimization version) is polynomially solvable for the class of outerplanar graphs.

We conclude the survey with a remark about the recognition problems for the classes of p-connected matrices with p=1,2. As

a consequence from the results in [Be79, Ag83], one can easily obtain a polynomial-time transformation from the SPLP to the set covering problem with the following property: if the matrix (d_{ij}) is p-connected (p=1,2,...) then so is the constraint matrix (a_{ij}). Hence all that is required is to develop polynomial-time recognition algorithms for p-connected binary matrices with p=1,2. In [BD79] Beresnev and Davydov present an $O(m^2n^2)$ algorithm for recognizing the 1-connected binary matrices. The question what the complexity of the recognition problem for the class of 2-connected binary matrices is appears to be more difficult and remains unsettled at the time of this writing.

References

[Ag83] A. A. Ageev, On complexity of minimization problems for polynomials in Boolean variables. Upravlayemye sistemy No. 23 (1983), 3-11 (in Russian).

[Ag89] A. A. Ageev, Graphs, matrices and the simple plant location problem. Upravlayemye sistemy No. 29 (1989), 3-12 (in Russian).

[BEW84] I. Bárány, J. Edmonds and L. Wolsey, Packing and covering with subtrees of a tree. Combinatorica 6 (1986), No 3, 221-233.

[BD62] R. E. Bellman and S. E. Dreyfus, Applied dynamic programming. Princeton University Press. Princeton, N.J., 1962.

[Be79] V. L. Beresnev, Minimization algorithms for polynomials in Boolean variables. Problemy kibernetiki 36 (1979),

225-246 (in Russian).

[BGD78] V.L.Beresnev, E.Kh.Gimadi and V.T.Dement'yev, Standardization extremal problems. Novosibirsk, Nauka, 1978, 333p. (in Russian).

[BD79] V.L.Beresnev and A.I.Davydov, On matrices with connectedness property. Upravlyaemye sistemy No.19 (1979), 3-13 (in Russian).

[Gi69] E.Kh.Gimadutdinov, On properties of solutions to a segment location problem. Upravlayemye sistemy No.2 (1969), 77-91 (in Russian).

[Gi83] E.Kh.Gimadi, Efficient algorithm for solving plant location problem with service regions connected with respect to an acyclic network. Upravlayemye sistemy, No.23 (1983), 12-23 (in Russian).

[Ko83] A.Kolen, Solving covering problems and the uncapacitated plant location problem on trees. Eur. J. Oper. Res 12 (1983), No.3, 266-278.

[Tr76] V.A.Trubin, Efficient algorithm for plant locating on trees. Doklady AN SSSR, v.231 (1976), No.3, 547-550 (in Russian).

A Genetic Algorithm for the Assembly Line Balancing Problem *

Edward J. Anderson† Michael C. Ferris‡

March 1990

Abstract

Randomized algorithms are being used extensively in optimization. We discuss one such algorithm, called a genetic algorithm, which can be used for combinatorial optimization. We will consider the application of the genetic algorithm to a particular problem, the Assembly Line Balancing Problem, and its implementation on a parallel architecture. The general scheme of a genetic algorithm is given, and its specialized use on our test bed problems is outlined. Although extensive parallel processing is available in these methods, the problems of communication and synchronization have not been considered in detail. We describe a prototype local neighborhood genetic algorithm for which communication is greatly reduced and give results of experimentation on several different neighborhood structures. A possible asynchronous scheme is also mentioned.

1 Introduction

Algorithms based on genetic ideas were first used to solve optimization problems more than twenty years ago (e.g. [Bag67]). During the 1970's this work continued, but was largely unknown. In the last five years, however, there has been increasing interest in genetic algorithms. There have been three conferences devoted to this topic and two books have appeared [Dav87, Gol89].

Many researchers have concentrated their efforts on nonlinear function optimization or the nonlinear programming problem. Our interest is in the application of genetic algorithms to combinatorial optimization problems. It is appropriate to start with an outline description of this type of approach to combinatorial optimization. A genetic algorithm (GA) works with a whole population of potential solutions, $(i = 1, \ldots, popsize)$, which we will call individuals. The population changes over time, but always has the same number of members. Each individual is usually represented by a single string of characters. At every iteration of the algorithm a fitness value, $f(i)$, is calculated for each of the current individuals. Based on this fitness function a number of individuals are selected as potential parents. Two new individuals can be obtained from two parents by choosing a random point along the string, splitting both strings at that point and then joining the front part of one parent to the back part of the other parent and vice versa. Thus parents A-B-C-A-B-C-A-B-C and A-A-B-B-C-C-C-B-A might produce offspring A-B-C-B-C-C-C-B-A and A-A-B-A-B-C-A-B-C when mated. This process is usually called crossover. Individuals may also change through random mutation when elements within a string are changed directly (normally this happens with only a low probability). The processes of crossover and mutation are collectively referred to as reproduction. The end result is a new population (the next "generation") and the whole process repeats. Over time this leads to convergence within a population with fewer and fewer differences between individuals. When a genetic algorithm works well the population converges to a good solution of the underlying optimization problem and the best individual in the population after many generations is likely to be close to the global optimum. A model algorithm is given below:

*This material is based on research supported by NSF Grant DCR–8521228 and Air Force Office of Scientific Research Grant AFOSR–89–0410

†University of Cambridge, Institute of Management Studies, Mill Lane, Cambridge CB2 1RX, England

‡Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706

Model Algorithm:

repeat
  for each individual i **do evaluate** f(i)
  for i=1 **to** (popsize/2) **do**
    select pairs of individuals j and k to mate based on their fitness
    reproduce using individuals j and k
until population variance is small

There are some similarities between a GA and the method of simulated annealing. Both approaches involve some random element in the way that the algorithm proceeds. In both cases the running time of the method will depend on certain parameter settings, with a greater likelihood that an optimal or near–optimal solution is found if the algorithm is allowed to run for a long time. Both methods have applicability across a wide range of problem domains – part of their attraction is that they hold out the promise of effectiveness without being dependent on a detailed knowledge of the problem domain. On the other hand there are substantial differences between the two approaches: for example since GAs operate using a whole population of individuals, they have a kind of natural parallelism not found in simulated annealing.

It is not easy to assess the effectiveness of this type of algorithm. For any particular problem there are likely to be special purpose techniques and heuristics which will outperform a more general purpose method. In a sense this may not be important. The value of simulated annealing, for example, is that for many problems half a day's programming and a day's computation will suffice to find as good a solution as would be obtainable after an half an hour's computation using a sophisticated method which might take three weeks or more to get working. Might something similar be true for genetic algorithms?

The paper consists of two parts. In the first part we have tried to establish that, for one particular type of problem, GAs have a clear advantage over the simplest of all generic approaches, which is the use of a neighborhood search technique with multiple starts. This is perhaps the minimal condition for genetic algorithms to be considered as a serious contender in solving hard combinatorial optimization problems. As far as we are aware this type of comparison has not been carried out before.

In the second part of the paper, we will describe more fully the parallelism which is inherent to the genetic algorithm and explain the various extensions which enable efficient implementation of the algorithm on message passing systems. Related work can be found in [GS89, Mä89, SG87, Tan87]

2 The Assembly Line Balancing Problem

We will look at the application of genetic algorithms to the Assembly Line Balancing Problem (ALBP). Suppose we wish to design a manufacturing line using a given number of stations, n. At each station someone will perform a number of operations on an item being made, before passing it on to the next station in the line. The problem is to assign the operations to the n stations in such a way as to produce a balanced line, given the time that each operation will take. The total output of the line will be determined by the slowest station, which is the station with the most work assigned to it. Our aim is to minimize the amount of work assigned to this station and thus to maximize the total throughput of the line. Thus far we have described a type of standard balancing problem – in a scheduling context this would be equivalent to minimizing makespan on parallel machines. The crucial feature of the ALBP, however, is that certain operations must be performed before others can be started. If we have this type of precedence relation between operations A and B, for example, then we cannot assign operation B to a station earlier than operation A (though both operations may be assigned to the same station).

The ALBP has attracted the attention of many researchers. Both heuristic and exact methods have been proposed for its solution. For a review of some of these methods see the papers [TPG86, Joh88]. Note that the ALBP is sometimes posed with the total operation time for each station constrained by some upper bound (the desired "cycle time") and the number of stations as the variable to be minimized. The ALBP is attractive as a test bed for GAs since there is a natural coding of a solution given by the station assignment for each operation. Also, though we do not expect a GA to be as effective as some of the special purpose heuristic methods, it will nevertheless be interesting to make comparisons between the two approaches.

There are a number of issues to be resolved in implementing a genetic algorithm for the ALBP. We will now deal with three of these issues; namely the coding scheme used, the method of calculating fitness and selecting individuals for mating, and the recombination mechanism (of mutation and crossover).

Coding

There are two aspects to the coding scheme for a genetic algorithm. One is the way that a solution is related to the elements of the string which codes for it. As we mentioned above a natural coding is available for the ALBP in which each operation is associated with a fixed position on the string and the code is simply the number of the station to which that operation is assigned. The other aspect of coding is the location of the operations on the string. This will be important since the crossover operation will be less likely to separate two pieces of information ("genes" in the genetic description) if they are close together on the string. We have chosen to put the operations into an order given by increasing numbers of predecessors, with ties broken by looking at the number of immediate predecessors.

Fitness and selection for the mating pool

At the heart of a GA is some calculation of fitness for each member of the population. This will determine how likely it is that an individual survives into the next generation, or is selected for mating. For the ALBP the fitness must include an element corresponding to the total time for the operations assigned to the slowest station. However we also wish to avoid solutions which are infeasible because of precedence constraints. Rather than rule these out directly, we will assign a large penalty cost to any of these infeasible solutions.

The fitness function we use is defined to be

$$exp(-k(Tmax + d * V + e * T2 + f * (Tmax - Tmin)))$$

Here $Tmax$ is the slowest station time, $T2$ is the second slowest station time, $Tmin$ is the fastest station time and V is the number of precedence violations. The constants d, e, f and k are chosen as follows. $d \gg 1$ in order that the precedence violations are removed as quickly as possible. e is chosen so that the second slowest time is taken into account (to decrease the solution value it would be valuable to have this time smaller than $Tmax$) but so that $Tmax$ dominates. The final term is added to try and force a balanced line (but with f chosen suitably small). The constant k is chosen so that the values of the fitness function lie within reasonable bounds.

We have also implemented a linear scaling of the fitness values. The scaling is performed in such a way that the average fitness remains constant but the maximum fitness is a multiple (usually 1.5) of this average value.

We have used two different methods to select individuals for mating. The first (stochastic sampling with replacement) selects each individual with a probability proportional to its fitness. Thus each time a selection is made individual i is chosen with probability given by

$$pselect_i = f_i / \sum_j f_j$$

The second method, which is usually called "remainder stochastic sampling without replacement" is similar but involves some selections being made in a deterministic way. First each individual is allocated samples according to the integer parts of $e_i = pselect_i * n$. The remaining samples are taken one by one, with at each stage the probability of individual i being selected proportional to the fractional part of e_i until the new population has the desired size. This scheme has the advantage that all above average individuals will survive to mate in the next generation. The second of these methods is generally acknowledged to be superior and this has been confirmed by numerical testing on the ALBP.

Recombination

There are two aspects to recombination: crossover and mutation. We have made some limited experiments with different crossover mechanisms with the aim of incorporating some problem specific knowledge. This has

been shown to be effective in some previous studies [Gre87]. One method that we tested was to concentrate the changes that crossover introduces within the slowest station. For each parent, this was achieved by changing some of the station assignments for operations currently assigned to one of the slowest stations to the station assignment that that operation has in the other parent. However this approach did not lead to any very substantial improvements over the more standard crossover mechanism described in the introduction. For our problem, a more natural scheme for crossover is to randomly generate an operation number and cross over that operation and operations which are its successors in the precedence graph. This appears to work slightly better than the scheme outlined in the introduction and is the scheme of choice for the remaining results reported here.

We have implemented two forms of mutation. The first is standard in the literature of genetic algorithms and involves the random change of a particular allocation of operation to station. We move a random operation from its station to the station immediately before or after it. The second scheme is motivated by our particular problem. Mutation is achieved by choosing two adjacent stations and moving each task from one of these stations to the other with small probability. We also tested concentrating the mutations within the stations achieving the slowest time. Over a number of runs this usually gave a slightly better solution, but the variance of the solutions was much greater and frequently infeasible solutions resulted.

3 Experimental Results (Comparison)

Experiments were performed on randomly generated problems having 40 operations; these are to be assigned to 6 stations. One factor which we varied was the number of precedence relations included. This can be conveniently measured as a "density" giving the number of precedence relations actually present or implied divided by the total number possible.

As stated earlier our aim is to compare an implementation of a genetic algorithm with a simple neighborhood search scheme. We have implemented the neighborhood search by taking a population of initial solutions and at each generation replacing each individual with a randomly generated neighbor. The randomly generated neighbor is given by applying the mutation operation described above to the individual and only replacing the current individual by its neighbor if the neighbor is as good as the individual. For the genetic algorithm we used a probability of crossover of 0.6 and a probability of mutation of 0.03 (for each element in a string). For both the neighborhood search method and the genetic algorithm we used a population size of 40.

We experimented with two kinds of starting solution. The first scheme generated the initial population entirely at random. In order that one can effectively program a genetic code quickly, this would be the method of preference for generating initial solutions. In this case, the genetic algorithm performed far better than the neighborhood scheme. Indeed, in most cases, the neighborhood scheme was unable to find a feasible solution. However, the genetic algorithm was able to find a feasible individual whose maximum time was within 10% of the optimal time for the problem in every case we tried.

However the situation is different when a preselected initial population is used. We generated a set of initial solutions using a method due to Arcus (see [Arc66]). This is extremely effective. In a significant proportion of cases the initial set of Arcus solutions contains at least one which is never improved upon by either the GA or neighborhood search method. In the other cases the best of the Arcus solutions is never far from the best solution found. Using the same parameters as before for the GA and with a limit of 40 iterations the performance of the GA and the neighborhood search scheme was comparable. It seems that there is premature convergence of the method around the few individuals which are generated by the Arcus scheme. With insufficient variability in the population the GA is unable to work well.

The total time assigned to the slowest station for these randomly generated problems was around 50. In cases where the GA or the neighborhood scheme did better it was usually by a margin of 1 unit, implying that the throughput of the line would be improved by about 2%.

There are a number of conclusions that can be drawn from these experiments. First we note how difficult it is to improve on the "standard" version of the genetic algorithm; the only area where changes were able to show some improvement for the problem we looked at was in the use of different mutation schemes. This characteristic of genetic algorithms is clearly beneficial if it is desired to use the technique on problem domains about which one has little knowledge. We have shown that for the assembly line balancing problem a genetic

algorithm performs significantly better than simple neighborhood search from a random initial population. The effectiveness of the method does depend somewhat on the density of the problem, the best results being obtained for problems with relatively low densities. If an initial population of good solutions is used then a genetic algorithm may offer little benefit in comparison with a more straightforward neighborhood search scheme. However, even in this case, both methods give an improvement of about 2% over the initial solutions after 40 generations.

4 Parallel Implementation Using Local Neighborhoods

Up to this point we have only been concerned with a comparison of the genetic algorithm with other schemes used to solve our problem. In the remainder of this paper we will look at the parallelism associated with the genetic algorithm initialized with a random population and try to improve on synchronization and communication costs. A GA would appear to be ideal for parallel architectures since evaluation and reproduction can be performed concurrently. In fact, for a large population of individuals, the use of many processors would seem enticing. However, this ignores the problem of communication and synchronization which are inherent in the selection mechanism described above. In this section we discuss how selection can be performed in a way which reduces the message passing required in a parallel architecture. Not only is this beneficial from the point of view of communication penalties but computational experience demonstrates that it will also improve the quality of solutions obtained.

Note that for both of the techniques of selection described previously, the processor effecting the selection needs to acquire the fitness values of every individual. This involves a large communication penalty for any message passing system.

Recently, several researchers have experimented with neighborhood schemes in which the fitness information need only be transmitted within the local neighborhood. The pioneers in this area are H. Mühlenbein and M. Gorges-Schleuter who have developed the ASPARAGOS system to implement an asynchronous parallel genetic algorithm [GS89, M̈89]. In this paper we will concentrate on two issues: first the determination of the best neighborhood structure to use, and second the difference between synchronous and asynchronous versions of the algorithm. We aim to clarify these issues by discussing the performance of a parallel GA on a particular problem.

A model algorithm for a scheme in which fitness information is only compared locally is as follows.

Local Neighborhood Algorithm:

repeat
  for each individual i **do**
    evaluate f(i)
    broadcast f(i) in the neighborhood of i
    receive f(j) for all individuals j in the neighborhood
    select individuals j and k to mate from neighborhood based on fitness
    request individuals j and k
    synchronize
    reproduce using individuals j and k
until population variance is small

For a parallel implementation, we assume that each individual in the population resides on a processor and communication is carried out by message passing. Our experimental results are concerned with several neighborhood schemes which we describe briefly now:

global: Here every individual is in the neighborhood of every other individual. The neighborhood size is the size of the population.

hypercube:

$$i \in \mathrm{nhd(j)} \iff d_H(i,j) \leq 1$$

where $d_H(i,j)$ is the number of different bits in the binary expansions of i and j. This can be viewed as each individual residing on the vertex of a hypercube with adjacent vertices giving its neighbors. The neighborhood size is one more than the dimension of the hypercube.

ring4:

$$i \in \text{nhd(j)} \iff |\text{i} - \text{j}| \leq 2$$

This can be viewed as each individual residing on a ring and its neighbors are those no further than two links away. The neighborhood size is four.

ring8:

$$i \in \text{nhd(j)} \iff |\text{i} - \text{j}| \leq 4$$

This can be viewed as each individual residing on a ring and its neighbors are those no further than four links away. The neighborhood size is eight.

grid4: Suppose $popsize = r^2$ and individuals are labeled as (u,v) with $u,v \in \{1,\ldots,r\}$. Let $d_r(a,b) := |(a \bmod r) - (b \bmod r)|$. Then

$$(u,v) \in \text{nhd((x,y))} \iff \text{d}_\text{r}(\text{u},\text{x}) + \text{d}_\text{r}(\text{v},\text{y}) \leq 1$$

This can be viewed as each individual lying on a grid and only communicating with the four grid points which differ by one in at most one component (with wrap around at the edges).

grid8: Using the same setup as grid4

$$(u,v) \in \text{nhd((x,y))} \iff \max\{\text{d}_\text{r}(\text{u},\text{x}), \text{d}_\text{r}(\text{v},\text{y})\} \leq 1$$

This can be viewed as each individual lying on a grid and only communicating with the eight grid points which differ by less than one in every component.

island: Suppose $popsize = r*s$ and individuals are labeled as (u,v) with $u \in \{1,\ldots,r\}$ and $v \in \{1,\ldots,s\}$. Then

$$(u,v) \in \text{nhd((x,y))} \iff \begin{cases} u = x \text{ and } |v-y| \leq 2 \text{ or} \\ v = y = 1 \text{ and } |u-x| \leq 1 \end{cases}$$

This can be viewed as r island populations which can only communicate with the rest of the world through a particular individual. The neighborhood size is between 4 and 6.

We now describe some further details of an implementation of this algorithm. Because we can no longer think of a single mating pool it is unclear how to implement remainder stochastic sampling without replacement. Instead we carry out selection using stochastic sampling with replacement. In the form given above we select two individuals j and k from the neighborhood based on fitness. In order to reduce communication it is possible to set $k = i$. This technique was used on our test problems with each of the neighborhood schemes above and performed at least as well and frequently better than the original scheme. For the rest of our experimentation, this technique was used. We postulate that (although the new technique allows potentially poor solutions to be involved in mating which could degrade performance) the greater variability in solutions considered is helpful.

Reproduction produces two offspring. Our strategy was to replace the current individual with its best offspring provided this offspring is better than the worst individual in the neighborhood. The question arises whether it is possible to use the other offspring? In order to answer this, the following techniques were tested:

noret: The less fit offspring is discarded.

retpar: The less fit offspring is sent to its other parent which it replaces if it is fitter than this parent.

retran: The less fit offspring is sent to a random neighbor which it replaces if it is fitter than this neighbor.

In the sequel, we shall refer to these techniques as "return policies".

5 Experimental Results (Parallelism)

The aim of these tests was to determine, if possible, which neighborhood scheme and which return policy was optimal. Although these results may depend on our particular problem, it is hoped that the conclusions of this research will be applicable in the general context of genetic algorithms applied to combinatorial

	optimality tolerance	noret	retpar	retran	twosel
global	0.1%	245	256	52	64
	1.0%	805	829	632	640
	2.0%	827	896	640	704
hypercube	0.1%	64	192	128	0
	1.0%	704	768	704	448
	2.0%	704	832	768	576
ring4	0.1%	197	216	85	128
	1.0%	826	790	768	448
	2.0%	831	804	832	640
ring8	0.1%	192	192	0	128
	1.0%	832	960	512	640
	2.0%	944	960	731	768
grid4	0.1%	128	64	256	0
	1.0%	832	576	704	448
	2.0%	832	640	768	576
grid8	0.1%	128	0	128	64
	1.0%	704	576	704	384
	2.0%	832	704	704	832
island	0.1%	349	309	355	64
	1.0%	829	854	768	640
	2.0%	829	876	916	704

Table 1: Number of individuals within optimality tolerance

optimization problems. Our experiments were confined to a randomly generated test set with the number of stations varying between 2 and 6, the number of operations varying between 40 and 50 and the density of the precedence graph ranging between 0.2 and 0.8. The operation times were generated either from a uniform distribution on [50,500] or a binomial distribution with parameters $n = 30$ and $p = 0.25$, The generator used to produce these problems is described in more detail in [TPG86]. Each problem was allowed to run for 400 generations on a population size of 64 starting from 5 random initial populations. Table 1 is a summary of the experimental results we obtained. For each of the neighborhood schemes we calculate how many individuals from the population at generation 400 are within 0.1%, 1.0% and 2.0% of the optimal solution for each of the problems and take their sum. We feel this is the significant figure for each of the procedures - it is an attempt to ascertain which neighborhood structure and which return policy works best, independent of other factors (such as density of precedence graph, number of operations, etc) on the problem. The maximum figure that could appear in the table is 1280.

In all cases, the minimum value found in the final generation by the genetic algorithm was within one-tenth of a percent of optimality. We do not consider the best solution found over all generations, since the communication involved in determining this solution would be enormous. However, in a serial implementation, the best solution found in the final generation was close to and in general equal to this value. Furthermore, the local neighborhood algorithm always outperformed the standard schemes where selection was performed (with or without replacement) on the entire population.

Note that the values reported in Table 1 are frequently multiples of 64. This is due to the fact that after 400 iterations, the GA has frequently converged and all the individuals have the same value. It is clear that different schemes will have different convergence rates. Some of the schemes above converged after around 200 generations and produced inferior quality solutions, mainly due to the fact that they did not look at enough different individuals. In order to obtain a fair comparison, we set the scale factor in the linear scaling of the fitness values to 1.2 instead of 1.5 (in the case of the hypercube or grid8 neighborhoods). Figures 1 and 2 show the effect of changing the scale factor. The graphs show the maximum, minimum and average values of the objective function for a particular problem instance averaged over 5 runs of the genetic algorithm as

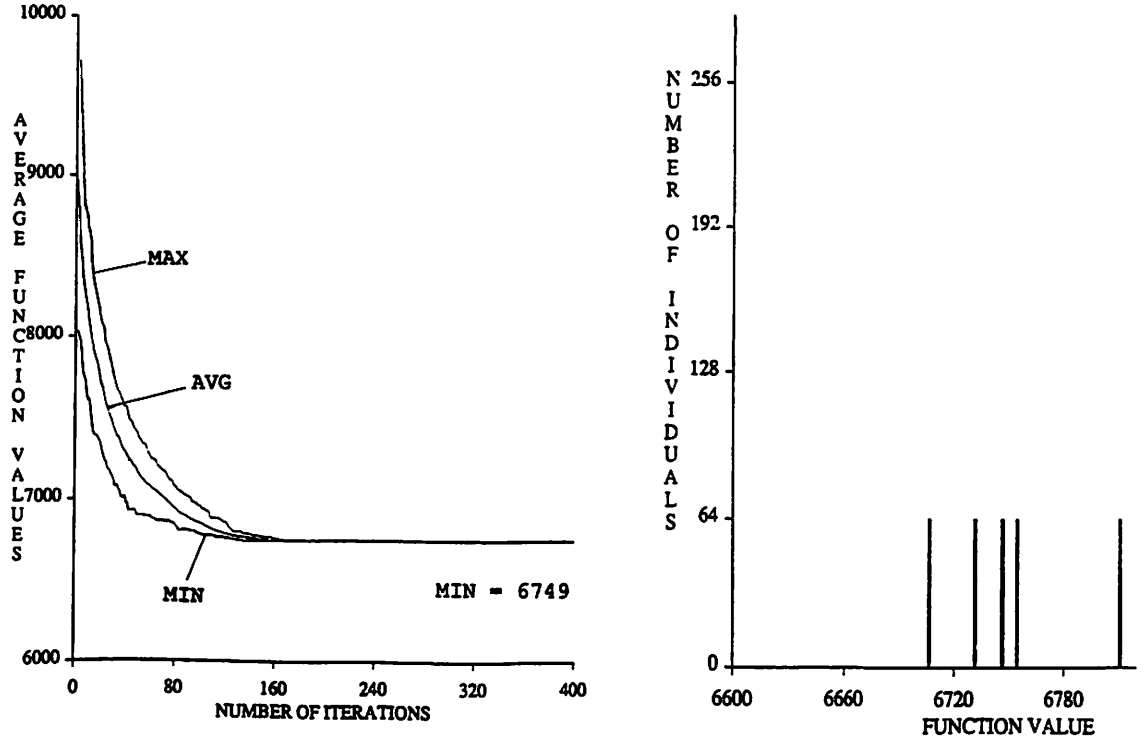


Figure 1: Scale factor 1.5, grid8 neighborhood

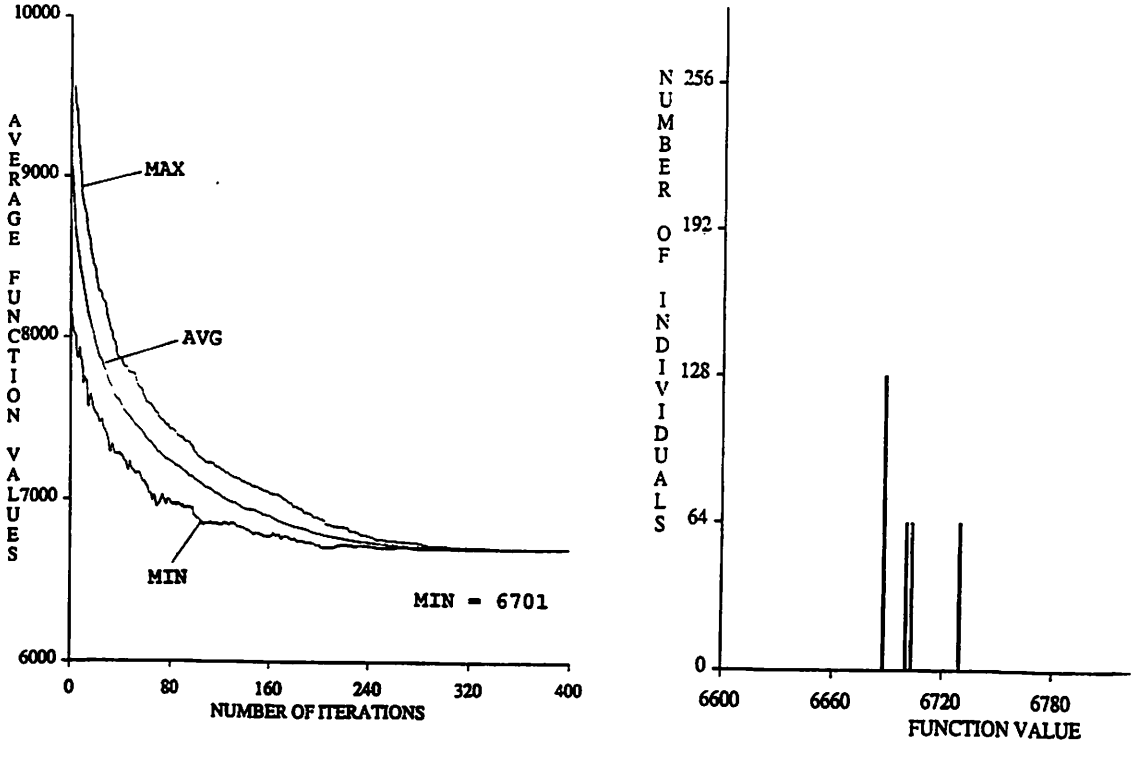


Figure 2: Scale factor 1.2, grid8 neighborhood

	global	hypercube	ring4	ring8	grid4	grid8	island
global	–	3:1	2:2	3:1	3:1	3:1	0:3*
hypercube	1:3	–	1:3	1:3	1:2*	2:2	0:4
ring4	2:2	3:1	–	2:2	3:1	3:1	1:3
ring8	1:3	3:1	2:2	–	3:1	3:1	1:3
grid4	1:3	2:1*	1:3	1:3	–	3:1	0:4
grid8	1:3	2:2	1:3	1:3	1:3	–	0:4
island	3:0*	4:0	3:1	3:1	4:0	4:0	–

Table 2: Neighborhood scheme rating

	noret	retpar	retran	twosel
noret	–	3:4	4:3	7:0
retpar	4:3	–	4:3	6:1
retran	3:4	3:4	–	4:3
twosel	0:7	1:6	3:4	–

Table 3: Return policy rating

a function of generation. In Figure 1, the linear scaling factor makes the maximum fitness 1.5 times the average fitness, whereas in Figure 2 the scale factor is 1.2. Notice that the convergence is slower in the second example, but the histograms (which are plots of the numbers of individuals against the maximum station time) show the quality of solutions is much better.

From Table 1 we produce two further tables which we claim show which neighborhood scheme is best and which return policies work well. Essentially, we order the triples found in Table 1. Two triples, (a, b, c) and (d, e, f) are ordered as follows:

$$(a,b,c) \text{ "is better than" } (d,e,f) \iff \begin{cases} d/a < 0.95 \\ \text{or } 0.95 \leq d/a \leq 1.05 \text{ and } e/b < 0.95 \\ \text{or } 0.95 \leq d/a \leq 1.05 \text{ and } 0.95 \leq e/b \leq 1.05 \text{ and } f/c < 0.95 \end{cases}$$

Essentially, one triple is better than another if is is lexicographically greater than the other (with two elements of the triple being treated as equal if they differ by less than 5%). To obtain the ijth entry of Table 2 we compare rows of Table 1. Thus entry (hypercube, ring8) in the table is 1:3 which means that for 1 return policy the hypercube neighborhood structure was better than the ring8 structure, whereas for the other 3 return policies, the ring8 structure gave better solutions. A * represents the fact that a tie occurred. From the results given in Table 2 we conclude that island is the best neighborhood scheme, followed by the global scheme and then the schemes ring4 and ring8. The grid4, grid8 and hypercube schemes are comparable to each other but not very good at all. We also note that the global scheme requires much more computation, since the neighborhood size is much bigger.

Table 3 gives the comparison of return policies by comparing columns of Table 1 in a similar manner as above. The results given in Table 3 are not as conclusive. In fact, noret, retpar and retran perform somewhat similarly. For completeness, we have included a column labeled twosel. This corresponds to selecting two individuals from the neighborhood and using policy noret for the worst offspring. As mentioned above, the results in the table show this does not perform as well as the other schemes and is clearly not a good policy to consider.

Since the communication costs associated with noret are smaller than those associated with retpar and retran and the quality of their solutions is very comparable, we conclude that discarding the less fit offspring is the best return policy and that the island neighborhood scheme (or a closely related variant of this) should be chosen as the neighborhood when these scale factors are used.

We also carried out further experimentation as to the effect of scaling on our results. Tables 4, 5 and 6

	optimality tolerance	noret	retpar	retran	twosel
global	0.1%	94	144	4	192
	1.0%	583	758	702	768
	2.0%	886	873	704	832
hypercube	0.1%	320	256	320	0
	1.0%	832	832	896	384
	2.0%	832	896	896	576
ring4	0.1%	425	370	168	384
	1.0%	866	965	873	960
	2.0%	941	1012	896	960
ring8	0.1%	299	376	448	128
	1.0%	768	960	960	576
	2.0%	896	960	960	832
grid4	0.1%	255	128	320	128
	1.0%	896	896	896	512
	2.0%	896	896	896	768
grid8	0.1%	256	256	192	64
	1.0%	896	896	832	768
	2.0%	1024	896	896	896
island	0.1%	108	145	378	256
	1.0%	865	793	810	576
	2.0%	879	896	960	768

Table 4: Number of individuals within optimality tolerance

are the corresponding results when we require the maximum fitness to be 1.15 times the average fitness.

The results given in Table 4 are better than those given in Table 1 as can be seen by a similar comparison as the one described above. Some of the conclusions we can draw from Tables 5 and 6 are slightly different from our previous conclusions. The striking difference is that the global scheme now performs badly. The ring4 and ring8 schemes are now the best, with the grid4, grid8 and island schemes being comparable to each other, but not as good. The global and hypercube schemes do not perform well. Furthermore, it appears that the best return policy is retran. However, this effect should be balanced against the extra communication cost.

A plausible explanation for the above behavior is that information is dissipated through the population much slower in the ring4, ring8 and island schemes, which allows many more local optima to be explored. The island scheme works better than the other schemes when the fitness values are not scaled down as much because it still allows several local optima to be explored, even if there is one dominant local optima.

	global	hypercube	ring4	ring8	grid4	grid8	island
global	–	2:2	1:3	1:3	1:3	1:3	0:4
hypercube	2:2	–	1:3	1:3	2:1*	2:2	2:2
ring4	3:1	3:1	–	3:1	3:1	3:1	3:1
ring8	3:1	3:1	1:3	–	4:0	4:0	3:1
grid4	3:1	1:2*	1:3	0:4	–	2:2	1:3
grid8	3:1	2:2	1:3	0:4	2:2	–	2:2
island	4:0	2:2	1:3	1:3	1:3	2:2	–

Table 5: Neighborhood scheme rating

	noret	retpar	retran	twosel
noret	–	4:2	2:4	5:1
retpar	2:4	–	2:4	4:2
retran	4:2	4:2	–	4:2
twosel	1:5	2:4	2:4	–

Table 6: Return policy rating

We conclude from this that an appropriate scaling of the fitness values will improve the quality of the solutions found. Clearly, experimentation has to be performed in order to calculate the correct scale factor for a given problem. Generally, increasing the scale factor speeds up convergence, and decreasing the scale factor gives better quality solutions, provided the genetic algorithm is allowed to run until it has converged.

6 Synchronization

The neighborhood schemes of the previous section effectively deal with communication penalties provided that care is taken to use a neighborhood structure which is appropriate for the machine architecture. The synchronization issue remains largely unsolved, but the following asynchronous scheme has proven very effective in practice.

Asynchronous Local Neighborhood Algorithm:

repeat
  for each individual i **do**
    evaluate f(i)
    broadcast f(i) in the neighborhood of i
    receive f(j) for all individuals j in the neighborhood
    select an individual j to mate from neighborhood based on fitness
    request individual j
    reproduce using individuals i and j
until population variance is small

Note that since we have removed the synchronization step, it may happen that we request an individual based on its fitness, but in fact receive an individual which has replaced the one requested. In practice, this does not seem to matter and in experiments carried out on our test examples, the above algorithm has produced results of comparable quality in somewhat smaller computational times.

7 Conclusions

The conclusions of this work are twofold. Firstly, the comparison of the standard genetic algorithm with a neighborhood search scheme with multiple restarts shows that the genetic algorithm outperforms this method and invariably produces better solutions.

We conclude from our experimental parallel work on the genetic algorithm that using a local neighborhood scheme is very important for achieving close to optimal solutions and in fact produces better solutions than the standard genetic algorithm. The choice of neighborhood seems to be machine dependent: our computational results indicate that a variant of the island or the ring schemes we define in this paper seems to give the best performance. If two offspring are produced from the reproduction phase, it is better to discard the less fit offspring. Mating on each processor should be between the individual that is situated at the processor and an individual selected from the neighborhood. Appropriate choice of scaling for the fitness values can improve the quality of solutions or speed up the convergence rate.

Further work on these techniques is in progress. New techniques for evaluation of fitness under constraints and applications to database query optimization are being developed.

Acknowledgement

The authors would like to thank Mr. Menglin Cao for his help in testing the algorithm.

References

[Arc66] A.L. Arcus. COMSOAL: A computer method of sequencing operations for assembly lines. In E.S. Buffa, editor, *Readings in Production and Operations Management.* John Wiley & Sons, New York, 1966.

[Bag67] J.D. Bagley. *The Behaviour of Adaptive Systems which employ Genetic and Correlation Algorithms.* PhD thesis, University of Michigan, 1967.

[Dav87] L. Davis, editor. *Genetic Algorithms and Simulated Annealing.* Pitman, London, 1987.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading MA, 1989.

[Gre87] J.J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In Davis [Dav87].

[GS89] M. Georges-Schleuter. Asparagos. In Schaeffer [Sch89], pages 416–421.

[Joh88] R. Johnson. Optimally balancing large assembly lines with FABLE. *Management Science*, 34:240–253, 1988.

[Mü89] H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In Schaeffer [Sch89], pages 416–421.

[Sch89] J.D. Schaeffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, California, 1989. Morgan Kaufmann Publishers, Inc.

[SG87] J.Y. Suh and D. Van Gucht. Distributed genetic algorithms. Technical Report 225, Computer Science Department, Indiana University, Bloomington, July 1987.

[Tan87] R. Tanese. Parallel genetic algorithms for a hypercube. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 177–183, Hillsdale, New Jersey, 1987. Lawrence Erlbaum Associates.

[TPG86] F.B. Talbot, J.H. Patterson, and W.V. Gehrlein. A comparative evaluation of heuristic line balancing techniques. *Management Science*, 32:430–454, 1986.

On Identifying in Polynomial Time Violated Subtour Elimination and Precedence Forcing Constraints for the Sequential Ordering Problem

N. Ascheuer[1], L. Escudero[2],
M. Groetschel[1], and M. Stoer[1]

[1] Universität Augsburg, F.R. Germany
[2] IBM T.J. Watson Research Center, Yorktwon Heights, NY

Abstract

In this paper we consider the so-called Sequential Ordering Problem (SOP) introduced in [4] that has a broad range of applications, mainly, in production planning for manufacturing systems. Given a set of nodes, the SOP consists of finding a Hamiltonian path, such that precedence relationships among the nodes are satisfied and a given linear function is minimized. In [4] an efficient inexact algorithm for obtaining feasible solutions is described. Here, we present a strong formulation of the problem and procedures for identifying subtour elimination constraints and precedence forcing constraints that are most violated by the optimal solution of a LP relaxation of the original problem. The complexity of these separation procedures is $O(n^4)$ and $O(rn^3)$, respectively, where n denotes the number of nodes and r is the number of precedence relationships.

1. Problem Definition

Let $\mathcal{H}$ denote a Hamiltonian path through a given set of nodes, say V. Let $i \rightarrow j$ mean that node i is immediately ordered before node j in a given $\mathcal{H}$. Let the acyclic directed graph (or digraph) $P = (V,R)$, where R is the set of directed arcs, such that $(i,j) \in R$ means that node i has to be ordered (immediately or not) before node j in any feasible $\mathcal{H}$. We clearly may assume that P is transitively closed.

Let the complete digraph $D_n = (V,A_n)$ and the matrix $C = \{c_{i,j}\}$ be such that $c_{i,j}$ for $(i,j) \in A_n$ gives the cost associated with $i \rightarrow j$. Define

$$\begin{aligned} \overleftarrow{R} &:= \{(j,i) \mid (i,j) \in R\} \\ \overrightarrow{R} &:= \{(i,k) \mid \exists j \text{ s.t. } (i,j),(j,k) \in R\} \end{aligned} \tag{1.1}$$

and let us define the digraph D=(V,A) by setting

$$A := A_n \setminus (\overrightarrow{R} \cup \overleftarrow{R}) \tag{1.2}$$

It is obvious that no feasible $\mathcal{H}$ contains an arc from $\vec{R} \bigcup \overleftarrow{R}$. The *Sequential Ordering Problem* (SOP) consists of finding a Hamiltonian path $\mathcal{H}$ with minimum weight in digraph D, such that the precedence relationships given by R are not violated.

The SOP has a broad application field. An obvious application is the ATSP with fixed city-origin and city-destination. The sequencing of the cities may require to satisfy some precedence relationships. Another typical application [6] is the sequencing of the operations' execution, mainly in scheduling manufacturing systems. See in [18] an interesting application and an inexact algorithm for the related STSP with precedence relationships.

The paper is organized as follows. Section 2 presents our favorite 0-1 model for the SOP. It also outlines the procedure for obtaining an optimal solution. It is based on an iterative tightening of the relaxations of the Subtour Elimination Constraints (SECs) and Precedence Forcing Constraints (PFCs). Sections 3 and 4 are devoted to the procedures for identifying violated SECs and PFCs, respectively. These procedures have complexity $O(n^4)$ and $O(rn^3)$, respectively, where $n \equiv |V|$ and $r \equiv |R|$. Finally, we offer some conclusions and outline future work.

2. The 0-1 Model

Let $x_{i,j}$ be a 0-1 variable such that $x_{i,j} = 1$ means that $i \rightarrow j$ (in a feasible $\mathcal{H}$) and, otherwise, it is zero. Since $x_{j,i} = 0$ must be satisfied for all $(j,i) \in \vec{R} \bigcup \overleftarrow{R}$ we can drop these variables. Incidentally, a key step in any practical implementation of this approach is the preprocessing procedure for reducing the cardinality of set A; i.e., arc (i,j) should be deleted from A if it has been detected that $i \not\rightarrow j$ in any feasible $\mathcal{H}$, or in any better solution than the incumbent one (if any). An efficient algorithm for the SOP preprocessing is described in [4].

There are several 0-1 models for the SOP; see, e.g., [1,4]. Our favorite 0-1 model is as follows

$$\min \; c^t x \tag{2.1}$$

subject to

$$x(A) \; = \; n - 1 \tag{2.2}$$

$$x(\delta^-(v)) \; \leq \; 1 \text{ for all } v \in V \tag{2.3}$$

$$x(\delta^+(v)) \; \leq \; 1 \text{ for all } v \in V \tag{2.4}$$

It is obvious that no feasible $\mathcal{H}$ contains an arc from $\vec{R} \bigcup \overleftarrow{R}$. The *Sequential Ordering Problem* (SOP) consists of finding a Hamiltonian path $\mathcal{H}$ with minimum weight in digraph D, such that the precedence relationships given by R are not violated.

The SOP has a broad application field. An obvious application is the ATSP with fixed city-origin and city-destination. The sequencing of the cities may require to satisfy some precedence relationships. Another typical application [6] is the sequencing of the operations' execution, mainly in scheduling manufacturing systems. See in [18] an interesting application and an inexact algorithm for the related STSP with precedence relationships.

The paper is organized as follows. Section 2 presents our favorite 0-1 model for the SOP. It also outlines the procedure for obtaining an optimal solution. It is based on an iterative tightening of the relaxations of the Subtour Elimination Constraints (SECs) and Precedence Forcing Constraints (PFCs). Sections 3 and 4 are devoted to the procedures for identifying violated SECs and PFCs, respectively. These procedures have complexity $O(n^4)$ and $O(rn^3)$, respectively, where $n \equiv |V|$ and $r \equiv |R|$. Finally, we offer some conclusions and outline future work.

2. The 0-1 Model

Let $x_{i,j}$ be a 0-1 variable such that $x_{i,j} = 1$ means that $i \rightarrow j$ (in a feasible $\mathcal{H}$) and, otherwise, it is zero. Since $x_{j,i} = 0$ must be satisfied for all $(j,i) \in \vec{R} \bigcup \overleftarrow{R}$ we can drop these variables. Incidentally, a key step in any practical implementation of this approach is the preprocessing procedure for reducing the cardinality of set A; i.e., arc (i,j) should be deleted from A if it has been detected that $i \not\rightarrow j$ in any feasible $\mathcal{H}$, or in any better solution than the incumbent one (if any). An efficient algorithm for the SOP preprocessing is described in [4].

There are several 0-1 models for the SOP; see, e.g., [1,4]. Our favorite 0-1 model is as follows

$$\min \; c^t x \tag{2.1}$$

subject to

$$x(A) \; = \; n - 1 \tag{2.2}$$

$$x(\delta^-(v)) \; \leq \; 1 \text{ for all } v \in V \tag{2.3}$$

$$x(\delta^+(v)) \; \leq \; 1 \text{ for all } v \in V \tag{2.4}$$

$$0 \; \leq \; x_{ij} \; \leq \; 1 \text{ for all } (i,j) \in A \tag{2.5}$$

$$0 \leq x_{ij} \leq 1 \text{ for all } (i,j) \in A \tag{2.5}$$

$$x(A(W)) \leq |W| - 1 \text{ for all } W \subset V, 2 \leq |W| \leq n-1 \tag{2.6}$$

$$x_{ij} \in \{0;1\} \text{ for all } (i,j) \in A \tag{2.7}$$

$$x((j{:}W)) + x(A(W)) + x((W{:}i)) \leq |W| \quad \text{for all } (j,i) \in \overleftarrow{R} \text{ and all } W \subset V \setminus \{i,j\},\ W \neq \emptyset \tag{2.8}$$

where

$$x(F) = \sum_{(i,j) \in F} x_{ij}\,, \quad \delta^-(v) = \{(w,v) \in A\}\,, \quad \delta^+(v) = \{(v,w) \in A\}\,,$$

$$A(W) = \{(i,j) \in A \mid i,j \in W\}\,, \quad (j{:}W) = \{(j,w) \in A \mid w \in W\} \text{ and } (W{:}i) = \{(w,i) \in A \mid w \in W.$$

We restrict c in (2.1) to the coordinates $A \subseteq A_n$. Constraint (2.2) defines the number of arcs in any $\mathcal{H}$. Constraints (2.3) (resp., (2.4)) prevent that more than one node is sequenced immediately before (resp., after) any other node. Constraints (2.6) are the Subtour Elimination Constraints SECs (see e.g. [14]). Constraints (2.8) are the Precedence Forcing Constraints PFCs.

Note that (2.1)-(2.7) can be easily converted into the classical Asymmetric Traveling Salesman Problem (ATSP). Let the AP-like LP problem (2.1)-(2.5) be named LPAP.

The basic methodology of the exact algorithm that we are using for solving (2.1)-(2.8) has the following main steps: (1) Obtaining an initial feasible solution. The inexact algorithm described in [4,5] can be used. (2) Optimizing a relaxation of the original problem. We start with LPAP. (3) Reduced cost based variable fixing coupled with an implication fixing analysis. We use the preprocessing procedure described in [4]. (4) Cutting plane generation by identification of a most violated SEC (Section 3) and PFC (Section 4). We also consider (see [1,10,12]) some types of lifted SEC's and PFC's. (5) The new constraints are added, and non-active constraints previously appended are deleted from the current LP relaxation. (6) The dual-based optimization of the new problem is performed. (7) If there is not any violated constraint except (2.7), a branch-and-cut phase is executed. We should emphasize the synergetic effect of combining the procedures for obtaining initial feasible solutions, performing reduced cost fixing and implications, and generating violated constraints. The overall framework of this methodology has been previously described and its computational results have been extensively analyzed in [3,6,11,13,14,16,17] among others. Elsewhere [1] we report

our provisional computational experience on some real-life problems. In [8] a different approach for getting strong lower bounds for the ATSP imbedded in the SOP is described. See related work in [2,7].

3. Identifying Violated Subtour Elimination Constraints

The problem of determining a violated subtour elimination constraint (2.6) is reduced (in the obvious way) to the same problem for the symmetric case. We outline here the procedure for completeness.

Let us assume that $\bar{x}$ is the optimal solution of problem (2.1)-(2.5) where some constraints (2.6) and (2.8) may have been appended. We now consider the values $\bar{x}_{i,j}$ as capacities of the arcs $(i,j) \in A$. The goal consists of identifying a node set $W \subset V$ such that the related constraint (2.6) is a most violated SEC (if any). Then, the set W will be

$$W = arg.max\{\bar{x}(A(W')) - |W'| + 1 > 0\} \text{ for all } W' \subset V, 2 \leq |W'| \leq n-1. \tag{3.1}$$

For this purpose we will construct a symmetric directed graph, say $\bar{D}$, for which a certain mincut problem is to be solved. But, first, let us introduce additional notation. Let

$$\bar{y}_v = \bar{x}(\delta^-(v)) + \bar{x}(\delta^+(v)) \text{ for all } v \in V. \tag{3.2}$$

Next, we define an (auxiliary) digraph, say $D_0 = (V_0, A_0)$, as follows:

$$V_0 = V \cup \{0\}, \text{ where 0 is a new node ,} \tag{3.3}$$

$$A_0 = A \cup \{ (0,v) \mid v \in V\} \cup \{(j,i) \mid (i,j) \in A \text{ and } (j,i) \notin A\}.$$

(That is, we add to V a node 0 (the "source"), and we add to A arcs $(0,v)$ for all $v \in V$ and all reverse arcs but do not create parallel arcs). Let us define the following capacities for D_0

$$c^0_{0,v} := 1 - \frac{1}{2}\bar{y}_v \text{ for all } v \in V \tag{3.4}$$

(Note that $c^0_{0,v} \geq 0$ for all $v \in V$, where $c^0_{0,v} = 0$ whenever the related constraints (2.3) and (2.4) are satisfied as equalities). Further, we set

$$c^0_{i,j} := c^0_{j,i} := \frac{1}{2}(\bar{x}_{i,j} + \bar{x}_{j,i}) \text{ for all } (i,j) \in A_0 \backslash \{(0,v) | v \in V\} \tag{3.5}$$

(If $(i,j) \notin A$ (resp., $(j,i) \notin A$) we set $\bar{x}_{i,j} := 0$ (resp., $\bar{x}_{j,i} := 0$).)

Let $\delta_0^-(W)$ denote the minimum capacity cut in $D_0 = (V_0, A_0)$ with respect to the capacities (3.4) and (3.5), such that $0 \notin W$, and let $c(\delta_0^-(W))$ denote the associated capacity. It follows from our construction that if

$$c(\delta_0^-(W)) \geq 1 \tag{3.6}$$

is satisfied then there is no SEC violated by the current LP solution $\bar{x}$. Otherwise, the SEC induced by W is a most violated constraint (2.6).

We now construct a symmetric directed graph, say $\bar{D} := (V_0, \bar{A})$, related to digraph $D_0 = (V_0, A_0)$, by setting

$$\bar{A} := A_0 \cup \{(v,0) | v \in V\} \tag{3.7}$$

with capacities

$$\bar{c}_{i,j} := \bar{c}_{j,i} := c^0_{i,j} = c^0_{j,i} \text{ for all } (i,j) \in A_0 \tag{3.8}$$

$$\bar{c}_{v,0} := c^0_{0,v} \text{ for all } v \in V \tag{3.9}$$

($\bar{D}$ is a symmetric digraph since if any two nodes, say u and v are linked by an arc then both arcs (u,v) and (v,u) are in the digraph). The underlying undirected graph $G = (V_0, E)$ of $\bar{D} = (V_0, \bar{A})$ is defined by

$$E = \{ ij \,|(i,j) \in \bar{A} \text{ and } (j,i) \in \bar{A}\} \tag{3.10}$$

with capacities

$$\hat{c}_{ij} := \bar{c}_{i,j} = \bar{c}_{j,i} \text{ for all } ij \in E \tag{3.11}$$

It has the property that for each $W \subset V$,

$$\hat{c}(\delta(W)) = c(\delta_0^-(W)) \tag{3.12}$$

where $\delta(W) = \{ ij \in E | i \in V \backslash W, j \in W\}$.

Thus, a minimum capacity cut $\delta(W)$ in G corresponds to a minimum capacity cut $\delta_0^-(W)$ and vice versa. Such a cut $\delta(W)$ can be obtained with the Gomory-Hu procedure. Padberg and Rinaldi [15] describe a practically efficient version of this procedure.

The main loop of the algorithm requires at most n major iterations. At any major iteration two nodes at least are "contracted" or "shrunk" into a a single node. Two main steps are executed at each major iteration: first, a series of tests are designed to shrink as many nodes as possible; and, second, a max-flow algorithm is executed by selecting any two nodes of the shrunk graph as nodes "source" and "sink". The algorithm described by Goldberg and Tarjan [9] is used in the second step; its complexity is $p(n) = O(mn \log(n^2/m))$, where $m = |E|$. The tests that are performed in the first step have a complexity not worse than $p(n)$. Then, the overall complexity is $O(n^4)$. Although this is not better than the complexity of other published algorithms, the "shrinking" mechanism suggests a better efficiency in practice; this is confirmed by extensive computational experience reported in [16,17]. Since the algorithm obtains successively a series of better cuts till an optimal one is reached, we may generate the related induced SECs if they are violated, instead of using only a 'best' one.

4. Identifying Violated Precedence Forcing Constraints

We now describe how one can decide whether a given point satisfies all precedence forcing constraints, and if it does not, how one can find a (most) violated PFC (2.8). Let us assume that a vector $\bar{x}$, for practical purposes the current LP-solution, is given. For every $(j,i) \in R$ we do the following. Construct a new digraph $D_{j,i} = (V_{j,i}, A_{j,i})$ from D=(V,A) by deleting some arcs (those which never appear in (2.8)) and by shrinking the nodes i and j. Formally, set

$$V_{j,i} = V \setminus \{i,j\} \cup \{v_{j,i}\} \tag{4.1}$$

where $v_{j,i}$ is a new 'special' node representing j and i, and

$$\begin{aligned} A_{j,i} = &\{(k,l) \mid (k,l) \in A,\ k \notin \{i,j\},\ l \notin \{i,j\}\} \\ &\cup \{(v_{j,i}, l) \mid (j,l) \in A,\ l \notin \{i,j\}\} \\ &\cup \{(k, v_{j,i}) \mid (k,i) \in A,\ k \notin \{i,j\}\} \end{aligned} \tag{4.2}$$

Figure 1 shows the digraph $D_{j,i}$ induced by shrinking the nodes i and j in digraph D. Note that we throw out every arc directed into j and every arc directed from i; by definition, $(j,i) \notin A$. Every arc $a \in A_{j,i}$ gets a capacity $\hat{x}_a$ that is nothing but the value $\bar{x}_a$ of the corresponding arc a in $D = (V,A)$. Note that a PFC (2.8) for the old digraph D can be written as

$$x(A_{j,i}(\hat{W})) \leq |\hat{W}| - 1 \tag{4.3}$$

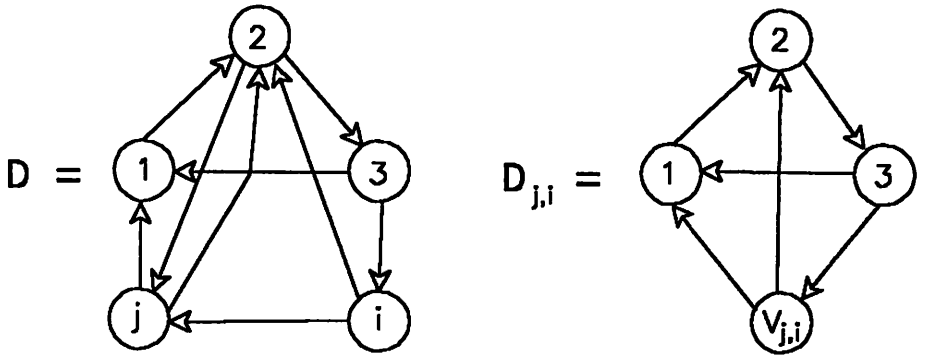


Figure 1. Original and shrinking graphs of precedence relationships

with respect to the new digraph $D_{j,i}$, where $\hat{W} = W \cup \{v_{j,i}\}$.

Our goal is to find a set $\hat{W}$ in $D_{j,i}$ with $v_{j,i} \in \hat{W}$ and $\hat{W} \geq 2$ such that the related constraint (4.3) forcing the precedence relation (i,j) is a most violated one or to show that no violated constraint of that type exists. To do this, we create an auxiliary digraph D_0 from $D_{j,i}$ (exactly in the same way as D_0 was constructed from D in Section 3) and determine a minimum capacity cut in D_0 that separates 0 from $v_{j,i}$. This can be done by applying any max-flow algorithm. Suppose $\delta_0^-(\hat{W})$ is such a cut with $0 \notin \hat{W}$ and $v_{j,i} \in \hat{W}$, and let $\gamma = c^{v_{j,i}}(\delta_0^-(\hat{W}))$ be its capacity. One can show that the following holds.

If $\gamma \geq 1$ then no PFC related to (i,j) is violated by $\bar{x}$. If $\gamma < 1$ then $\hat{W} \geq 2$ and letting $W^*: = \hat{W} \backslash \{v_{j,i}\}$ the PFC

$$x((j:W^*)) + x(A(W^*)) + x((W^*:i)) \leq |W^*| \tag{4.4}$$

is violated by $\bar{x}$, in fact, this is a most violated constraint forcing the precedence relation (i,j).

For obtaining a minimum capacity cut $\delta_0^-(\hat{W})$ separating 0 from $v_{j,i}$ we use the algorithm described in [9]; its complexity is $O(n^3)$.

For illustrative purposes let us consider the complete digraph $D_n = (V, A_n)$ for $n = 7$ and the matrix C shown in Table 1. The set R will be $R = \{(1,j)\} \cup \{(i,5)\}$ for $j = 4,5,6,7$ and $i = 1,4,6,7$. Our inexact algorithm [4] gives the feasible solution $1 \rightarrow 4 \rightarrow 2 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 3$. The value of the objective function is $\bar{z} = 21.25$. The LPAP optimal value is $\underline{z} = 18.00$ and, then, there is a 14.11% gap. The current implementation of our (exact) algorithm adds 8 cuts. The strongest LP lower bound is $\underline{z} = 20.75$ (then the gap is 2.29%). A branch-and-cut phase requires two nodes to prove the optimality of the initial solution. We currently have a PC and a mainframe implementation of the algorithm described above. The PC-version solves problems up to $n = 100$ nodes and $r = 280$ precedence relationships in less than two and a half CPU hours. The mainframe version does this in a few seconds.

\n2 \ n1\	1	2	3	4	5	6	7
1	-	1.00	2.00	0.75	0.00	3.00	1.00
2	4.00	-	5.00	3.25	4.00	6.00	0.00
3	7.00	8.00	-	5.50	7.00	9.00	8.00
4	2.75	2.50	2.25	-	2.75	5.25	2.50
5	0.00	1.00	2.00	0.75	-	3.00	1.00
6	10.00	11.00	12.00	10.75	10.00	-	11.00
7	4.00	0.00	5.00	3.25	4.00	6.00	-

Table 1. Matrix C

Conclusions

In this work we have presented a 0-1 model for the Sequential Ordering Problem. It is stronger than the model introduced in [4,5]. We have also outlined an (exact) LP-based algorithm. An extensive computational study is in progress. More theoretical work is required mainly for identifying in reasonable polynomial time violated lifted SECs and PFCs. In any case, we believe that the LP-based approach is a most promising way to get strong lower bounds on optimal solutions for this type of problems.

Acknowledgements

We appreciate Manfred Padberg's and Giovanni Rinaldi's comments.

References

[1] N. Ascheuer, L.F. Escudero, M. Groetschel, and M. Stoer, On LP bounds for the sequential ordering problem (in preparation).

[2] E. Balas and M. Fischetti, A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets, MSRR-556, GSIA, Carnegie Mellon University, Pittsburg, PA.

[3] H. Crowder and M. Padberg, Solving large-scale symmetric traveling salesman problems to optimality, **Management Sciences** 26 (1980) 495-509.

[4] L.F. Escudero, An inexact algorithm for the sequential ordering problem, **European J. of Operational Research** 37 (1988) 236-253.

[5] L.F. Escudero, On the implementation of an algorithm for improving a solution to the sequential ordering problem, **Trabajos de Investigacion-Operativa** 3 (1988) 117-140.

[6] L.F. Escudero, A production planning problem in FMS, **Annals of Operations Research** 17 (1989) 69-104.

[7] M. Fischetti, Facets of the asymmetric traveling salesman polytope, **Mathematics of Operations Research** (1989, to appear).

[8] M. Fischetti and P. Toth, An additive bounding procedure for combinatorial optimization problems, **Operations Research** 37 (1989) 319-328.

[9] A.V. Goldberg and R.E. Tarjan, A new approach to the maximum flow problem, **Journal of ACM** 35 (1988) 921-940.

[10] M. Groetschel, Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme (Hain, Meisenheim am Glan, 1977).

[11] M. Groetschel, M. Juenger and G. Reinelt, A cutting plane algorithm for the linear ordering problem, **Operations Research** 34 (1984) 1195-1220.

[12] M. Groetschel and M. Padberg, Lineare Charakterisierungen von Travelling Salesman Problemen, **Z. Operations Research** 21 (1977) 33-64.

[13] K. Hoffman and M. Padberg, LP-based combinatorial problem solving, **Annals of Operations Research** 5 (1986) 145-194.

[14] M. Padberg and M. Groetschel, Polyhedral computations, in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy-Kan and D.B. Shmoys (eds.), **The Traveling Salesman Problem, A guided tour of combinatorial optimization** (Wiley, NY, 1985) 251-360.

[15] M. Padberg and G. Rinaldi, On an efficient algorithm for the minimum capacity cut problem, Preprint, New York University, NY, 1987.

[16] M. Padberg and G. Rinaldi, Optimization of a 532-city symmetric traveling salesman problem, **Operations Research Letters** 6 (1987) 1-7.

[17] M. Padberg and G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, num. 319, Laboratoire d'Econometrie de l'Ecole Polytechnique, Paris, 1989.

[18] W. Pulleyblank and M. Timlin, Precedence constrained routing, ORSA/TIMS Joint National Meeting, New York, 1989.

FACTORING CARTESIAN-PRODUCT GRAPHS AT LOGARITHMIC COST PER EDGE

F. Aurenhammer, J. Hagauer* and W. Imrich†

April 6, 1990

Abstract

Let G be a connected graph with n vertices and m edges. We develop an algorithm that finds the prime factors of G with respect to Cartesian multiplication in $O(m \log n)$ time and $O(m)$ space. This shows that factoring G is at most as costly as sorting its edges. The algorithm gains its efficiency and practicality form using only basic properties of product graphs and simple data structures.

1 Introduction.

The Cartesian product is one of several standard products of graphs. Sabidussi [7] was the first to show that all finite connected graphs and a large class of infinite connected graphs have unique prime factorizations with respect to Cartesian multiplication. Idependently, Vizing [9] gave a proof of the unique prime factorization of finite connected graphs.

*Institutes for Information Processing Graz, Technical University of Graz, Austria

†Department of Mathematics and Applied Geometry, Montanuniversität Leoben, Austria

Sabiduss's proof is non-algorithmic. He uses a tower of successively coarser equivalence relations on the edges set E of the graph $G(V, E)$ under question. The prime factors of G then correspond to the equivalence classes in the coarset relation. By exploring the structure and size of these relations Feigenbaum et al. [3] were able to compute the prime factors of G in $O(n^4\sqrt{n})$ time, for $n =\mid V \mid$.

Independently, Winkler [10] published a polynomial-time algorithm by applying a method developed in a joint paper with Graham [5] to canonically embed G into a Cartesian product. The embedding is achieved by determining the equivalence classes of a particular relation $\hat{\Theta}$ on E. Unique prime factorization is then obtained by making this embedding surjective by unifying some of the equivalence classes of $\hat{\Theta}$. The implementation of this method, as described in [2], runs in time $O(n^4)$. Recent improvements in the computation of $\hat{\Theta}$ and in the second part of Winkler's algorithm (see [1] and [6], respectively) lead to a time complexity $O(mn + n^2 log^2 n)$ (and space complexity $O(n^2)$), for $m =\mid E \mid$.

In some respects, of course, much more than just the prime factorization of a connected graph G is computed in the approaches above. So it appears natural to try to find the prime factor decomposition of G via computing the (coarest) Sabidussi relation directly in order to further improve the complexity. This idea is pursued in the present paper. It leads to an algorithm requiring $O(m \log n)$ time and optimal $O(m)$ space. Since $n - 1 \leq m \leq \binom{n}{2}$ this is no more than the complexity of sorting the m edges of G. We conjecture the time complexity to be optimal too, but currently cannot show it.

The new algorithm partially gains its efficiency from using only basic properties of Cartesian product graphs. Section 2 is devoted to a brief discussion of such properties. The skeleton of the algorithm and a proof of its correctness are given in Section 3. More implementation details and the complexity analysis are provided in Section 4.

2 Cartesian product graphs.

Let $G(V, E)$ be a finite connected graph; notations V and E stand for the vertex set and edge set of G, respectively. We only consider undirected graphs without loops or multiple edges, so an edge can be viewed as in unordered pair (x, y) of distinct vertices. We start by defining Cartesian multiplication for graphs and continue with a review of basic properties of product graphs with regard to their relevance for the factorization algorithm in Section 3.

The *Cartesian product* $G_1 \Box G_2$ of two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ is definded as the graph $G(V, E)$, where

$$V = V_1 \times V_2 = \{\binom{u}{v} \mid u \in V_1, v \in V_2\}$$

and

$$E = \{(\binom{u}{v}, \binom{u'}{v'}) \mid (u, u') \in E \quad \text{and} \quad v = v', \quad \text{or} \quad u = u' \quad \text{and} \quad (v, v') \in E_2\}.$$

Intuitively speaking, by multiplying G_1 and G_2 each vertex of G_1 is replaced by a copy of G_2, and corresponding vertices of copies of G_2 are connected by extra edges. The operator $\Box$ indicates that the product of two single edges is a cycle of length four, that is, $K_2 \Box K_2 = C_4$. $\Box$ is commutative and associative in an obvious way and has the one-vertex graph K_1 as a unit. A graph G is called prime if $G = G_1 \Box G_2$ implies $G_1 = K_1$ or $G_2 = K_1$. A set $\{G_1, \ldots, G_t\}$ of graphs is called a *prime factorization* of G if

$$G = \Box_{i=1}^{t} G_i, \quad G_i \text{ prime, and } G_i \neq K_1, \quad \text{for } 1 \neq i \neq t.$$

We have already mentioned that every finite connected graph G admits a unique prime factorization (up to order and isomorphism of the factors); see [7], [9]. The elements of this factorization are connected graphs and are called *prime factors* of G. Now, let $G_1, \ldots, G_t$ be connected, prime graphs different from K_1 and let $G(V, E) = \Box_{i=1}^{t} G_i(V_i, E_i)$. Below we shall derive several more or less obvious, though important properties of G. By definition of G we have

$$V = V_1 \times \ldots \times V_t, \text{ and } E = \{((v_1, \ldots, v_t), (w_1, \ldots, w_t))\},$$

where $v_i, w_i \in V_i$ for $1 \leq i \leq t$ and $(v_j, w_j) \in E_j$ for some j but $v_k = w_k$ for $k \neq j$. Let us define an equivalence relation on E that reflects the prime factorization of G in the way that edges of the same prime factor belong to the same equivalence class. Two edges $e, e' \in E, e = ((v_1, \ldots, v_t), (w_1, \ldots, w_t))$, $e' = ((v'_1, \ldots, v'_t), (w'_1, \ldots, w'_t))$ are *equivalent*, in symbols $e \sim e'$, if $v_i \neq w_i$ if and only if $v'_i \neq w'_i$ for some $i, 1 \leq i \leq t$. Note that, by definition of $e, v_i \neq w_i$ occurs for exactly one i. Clearly $\sim$ is an equivalence relation. Since G_i is prime and $G_i \neq K_1$ for each $i, 1 \leq i \leq t$, we have index $(\sim) = t$ i.e. t is the number of equivalence classes of $\sim$. In fact, $\sim$ is the relation Sabidussi used in his paper [7] when he proved the uniqueness of the factorization. In order to find the prime factorization $\{G_1, \ldots, G_t\}$ of a given graph G this relation must be computed. More specifically, we need to calculate the bijection defined below.

Let $n_i = \mid V_i \mid$, for $1 \leq i \leq t$. For each i, label the vertices of G_i from 1 to n_i. Without loss of generality, these vertices are identified with their labels. Thus each vertex v of G corresponds to a t-tuple of integers. Let $N_i = \{1, \ldots, n_i\}$ and let $i_1, \ldots, i_t$ be a permutation of $1, \ldots, t$. The algorithm we are going to present calculates the bijection Π: $V \rightarrow N_{i_1} \times \ldots N_{i_t}$ such that $G \simeq \Box_{k=1}^{t} G'_k$, where G'_k are subgraphs of G induced by the vertices $v \in G$ with $\Pi\ (v) = (v_1, \ldots, v_t)$, $v_j = 1$ for all $j \neq k, 1 \leq j \leq t$, and furthermore $G'_k \simeq G_{ik}$. ($\simeq$ denotes isomorphism.) It is obvious that $\sim$ can be computed easily once Π is available. Note that t, the index of $\sim$, equals the number of coordinates of each vertex of G and is unknown when the algorithm starts. Below are some more properties of $\sim$.

Lemma 1: Each vertex of G is incident with at least one representative of each equivalence class of $\sim$.

The proof directly follows from the definitions of $\sim$ and from $G_i \neq K_1$, for $1 \leq i \leq t$. In particular, the lemma implies

$$t \leq \min \{p(v) \mid v \in V\},$$

with $\rho(v)$ denoting the degree of v. The following assertion strongly reflects the inherent structure of the Cartesian-product graph G. Again, the proof is easy and is left to the reader.

Lemma 2: Let $H(V_H, E_H)$ be a connected subgraph of G such that all edges in E_H are equivalent, and consider some edge $(v, w) \in E$ with $v \in V_H, w \notin V_H$. If (v, w) is not equivalent to the edges in E_H then there is a subgraph $L(V_L, E_L)$ of G such that
(1) $B = \{(x, y) \in E \mid x \in V_H, y \in V_L\}$ realizes an isomorphism between H and L.
(2) $(v, w) \in B$, and all edges in B are equivalent.

Taking a single edge of G for H yields the following corrollary:

Lemma 3: For any two adjacent edges $e, e' \in E$ with $e \not\sim e'$ there is exactly one chordless square in G that contains e and e' and where opposite sides are equivalent.

3 The factorization algorithm.

Our algorithm for factoring a connected input graph $G(V, E)$ computes the Sabidussi relation $\sim$ on E via the bijection Π defined earlier. The major part clearly is the calculation of Π where the algorithm roughly proceeds as follows.

Starting with a designated vertex $v_0 \in V$, the vertices of G are visited in BFS (breadth first search) order. For the set V' of vertices visited so far a bijection $c : V' \to \mathrm{N} \times \ldots \times \mathrm{N}$ (t times) is maintained. Initially $t = \rho(v_0)$ according to Lemma 1. We call $c(v)$ the (current) *coordinate vector* of v and write $c_i(v) = k$ if the i-th component of $c(v)$ equals k, for $1 \leq i \leq t$. The heart of the algorithm is a repeated application of Lemma 2 and its corrollary. Using the coordinate vectors of the vertices, the existence of the subgraphs L defined there can be decided efficiently. In case of non-existence, the number t of coordinates is reduced. Finally, the algorithm ends with $c = \Pi$, and thus with a partition of E into the equivalence classed with respect to $\sim$.

3.1 The overall structure.

Some notation first for the description of the algorithm. For each $v \in V$, let level(v) denote the number of edges of a shortest path in G between v_0 and v, i.e. $d(v_0, v)$. The edges $(v, w) \in E$ incident with v are called *up-edges*, *down-edges*, and *cross-edges* according to whether level(v) − level(w) is -1, $+1$, or 0, respectively. v is called *coordinized* if $c(v)$ has been assigned, and v is called *saturated* if all its adjacent vertices are coordinized. Among others, the following conditions are kept invariant during the execution of the algorithm.

(1) If v is saturated then all vertices w with level(w) $\leq$ level(v) are coordinized.

(2) Let (u, v) be an edge between two coordinized vertices at least one of which is saturated. Then $c(u)$ and $c(v)$ differ in exactly one component.

The informal description of the algorithm given below falls back upon two subroutines. The SQUARE TEST for a set of edges essentially refers to Lemma 2 and is specified later. REDUCE $(v_1, \ldots, v_k)$ replaces – for all coordinized vertices u – the subvector of $c(u)$ defined by the indices of components in which at least one of $c(v_1), \ldots, c(v_k)$ differs from $c(v_0)$, by a new single component. Thereby, two subvectors are replaced by the same component if and only if they have been equal themselves.

Algorithm FACTOR:

Initially, coordinize v_0 and its adjacent vertices $v_1, \ldots, v_t$ by setting $c_i(v_0) = 1$ and $c_i(v_i) = 2$ for $i = 1, \ldots, t$, and $c_j(v_i) = 1$ for $i, j = 1, \ldots, t$ and $j \neq i$. (This saturates v_0.) Suppose we have already saturated a set of vertices. Let v be the next vertex in BFS order with respect to v_0 (v is already coordinized.) Saturate v as follows.

Step 1: Apply the SQUARE TEST to the set of all cross-edges of v.

Step 2: For all up-edges (v, u) of v do: (v's down-edges have been scanned when v was coordinized.) If u is coordinized then no action is taken. ((u, v) is a down-edge of a coordinized vertex, u.) Otherwise coordinize u as follows:

Case 2.1. There is no down-edge of u but (u,v). Apply REDUCE (v). Then there is a unique index j with $c_j(v) \neq 1$. Coordinize u with

$$\begin{cases} c_i(u) &= 1, \text{ for all } i \neq j, \text{ and} \\ c_j(u) &= 1 + \max\{c_j(x) \mid x \text{ coordinized}\}. \end{cases}$$

Case 2.2. There are down-edges $(u,w_1),\ldots,(u,w_k)$ different from (u,v).

Case 2.2.1. If $c(v)$ differs from $c(w_i)$ in more than 2 components for at least one $i, 1 \leq i \leq k$, then REDUCE $(w_1,\ldots,w_k,v)$ and coordinize u as in Case 2.1.

Case 2.2.2. If $c(v)$ differs from $c(w_i)$ in exactly one component for each $i, 1 \leq i \leq k$, then REDUCE $(w_1,\ldots,w_k,v)$ and coordinize u as in Case 2.1.

Case 2.2.3. There is at least one down-edge (u,w) such that $c(w)$ and $c(v)$ differ in exactly 2 components, the i-th and the j-th, say. Test whether there are down-edges (v,x) of v and (w,x) of w such that $c(x)$ and $c(v)$ differ, say, in the i-th, and $c(x)$ and $c(w)$ differ in the j-th. If not then REDUCE $(w_1,\ldots,w_k,v)$ and coordinize u as in Case 2.1. (Searching for the existence of such an x will later be referred to as DOWN TEST.)

If yes, then consider the vector

$$c = (c_1(v),\ldots,c_{j-1}(v),c_j(w),\ldots,c_t(w)).$$

If c has already been assigned to some vertex y such that (v,y) is an up-edge of v then REDUCE $(w_1,\ldots,w_k,v)$ and coordinize u as in Case 2.1. (Searching for the existence of such a y will later be referred to as EQUALITY TEST.)
Assign $c(u) = c$, otherwise. (This is the only case where u is coordinized with more than one component differing from 1.)
Now u is coordinized. Apply the SQUARE TEST to the set of all down-edges of u.

Step 3: (Now v is saturated.) Apply the SQUARE TEST to the set of all up-edges of v.

In order to complete the description of the overall structure of Algorithm FACTOR, the SQUARE TEST needs to be explained. We restrict attention to down-edges since the test is performed in analogous manner for cross- and up-edges. Let $D_u = \{(u, w_1), \ldots, (u, w_k)\}$ be the set of down-edges of the vertex u.
SQUARE TEST (for D_u):

Case S1: There is some $(u, w) \in D_u$ such that $c(w)$ differs from $c(u)$ in more than one component. REDUCE $(u, w_1, \ldots, w_k)$ and return.

Case S2: ($c(w)$ and $c(u)$ differ in exactly one component for all $(u, w) \in D_u$.)

Case S2.1: Differences occur in one and the same component for all $(u, w) \in D_u$. Return.

Case S2.2: There are $(u, w), (u, w') \in D_u$ such that $c(w)$ ($c(w')$) differs from $c(u)$ in the i-th (j-th) component, for $i \neq j$. Test the two conditions below.

(a) There is some $(u, x) \in D_u, c_r(x) \neq c_r(u)$ for some $r \neq i$ iff there is a down-edge (w, y) of $w, c_r(y) \neq c_r(w), c_r(y) = c_r(x)$.

(b) There is some $(u, x) \in D_u, c_i(x) \neq c_i(u)$ iff there is a down-edge (w', y) of $w', c_i(y) \neq c_i(w'), c_i(y) = c_i(x)$.

If one of (a) or (b) is not satisfied then REDUCE $(u, w_1, \ldots, w_k)$ and return. (Testing for the validity of (a) and (b) will later be referred to as EDGE TEST.)

3.2 Correctness arguments.

In order to ease the argumentation about the correctness of Algorithm FACTOR, each edge of G scanned so far is attributed a unique color. Let $(u, v) \in E$ such that u and v are coordinized vertices at least one of which is saturated. Then $c(u)$ and $c(v)$ differ in the i-th component for exactly one $i, 1 \leq i \leq t$, and i is said to be the *color* of (u, v). We are going to show that, for all $e, e' \in E$, $e \sim e'$ holds if and only if e and e' have the same color after termination of the algorithm.

Lemma 4: Whenever (during the execution of Algorithm FACTOR) e and e' have the same color then $e \sim e'$.

Proof: The claim is true after initialization since distinct edges of v_0 have different colors. By induction on coordinized vertices we prove that, whenever a color is assigned to an edge the first time, and whenever an edge is recolored in a REDUCE-step, the claim is true. Let us review the actions taken by the algorithm during the saturation of vertex v.

Case 2.1: There is no down-edge of u but (u,v). So, for no down-edge (v,x) of v there is a square containing (u,v) and (v,x). Hence, Lemma 3 implies $(u,v) \sim (v,x)$. Accordingly, REDUCE (v) identifies the colors of all down-edges of v, and u is coordinized such that (u,v) is given this unique color.

Case 2.2.1: Assume that there is a down-edge (u,w) of u such that $c(w)$ and $c(v)$ differ in more than 2 components. Then, for no down-edge (v,x) of v, the edge (x,w) can exist since $c(w)$ and $c(v)$ would differ in at most 2 components, otherwise. So, no square containing (u,v) and (u,w) exists, that is, $(u,v) \sim (u,w)$. Assume now the existence of a down-edge (u,u') of u with $(u,u') \not\sim (u,v)$. Then, by Lemma 2, there are vertices v' and w' with up-edges $(v',u'),(v',v)$ and $(w',u'),(w',w)$ respectively, such that $(v',u') \sim (w',u') \sim (v,u)$ and $(v',v) \sim (w',w) \sim (u,u')$. Hence $c(u')$ and $c(w')$ must differ in exactly 1 component and therefore $c(v)$ and $c(w)$ in at most 2 which contradicts Case 2.2.1 and proves $(u,v) \sim (u,w_i)$ for all down-edges $(u,w_i), w_i \neq v$. As a consequence, all down-edges of v and $w_1,\ldots,w_k$ are equivalent, too.
This justifies the color identification in REDUCE $(w_1,\ldots,w_k,v)$ as well as the coloring of (u,v).

Case 2.2.2: For all down-edges (u,w_i) of $u, w_i \neq v, c(w_i)$ and $c(v)$ differ in exactly one component. Assume first that there is a down-edge (v,x) of v and an up-edge (x,w_i) of x for some i. Since $c(w_i)$ and $c(v)$ differ in exactly one component we have $(x,w_i) \sim (v,x)$. Moreover, since u forms a square with these two edges, $(u,w_i) \sim (u,v)$. If, on the other hand, x does not exist then $(u,w_i) \sim (u,v)$ again. Both assertions follow from Lemma 3. We conclude again that all down-edges of u are equivalent and complete the argumentation as in Case 2.2.1.

Case 2.2.3: $c(v)$ and $c(w)$ differ in exactly 2 components. If the down-edges (v,x) and (w,x) do not exist then $(u,v) \sim (u,w)$ by Lemma 3.

Assume next that (v,x) and (w,x) do exist, but the resulting vector c has already been assigned to some vertex y. Since up to now all coordinate vectors are different and since y has passed the SQUARE TEST for down-edges, there must be down-edges (y,v) and (y,w) of y. Thus, there is a subgraph G' with vertices u,v,w,x,y and edges $(u,v),(u,w),(v,x)(w,x),(y,v),(y,w)$. However, all edges of G' are equivalent. (This easily follows from Lemma 3.) In particular, $(u,v) \sim (u,w)$ again. As in Case 2.2.1, assuming the existence of some other down-edge (u,u') of u with $(u,u') \not\sim (u,v)$ results in a contradiction (by Lemma 2). So all down-edges of u are equivalent and the argument is completed as in Case 2.2.1.
Finally, assume that $c \neq c(x)$, for all vertices x coordinized so far. When coordinizing u with c, (u,v) gets the color of (w,x) and (u,w) gets the color of (v,x) which is in accordance with Lemma 3. Since these two colors currently are different it is irrelevant whether $(u,v) \sim (u,w)$ or not.
SQUARE TEST (for down-edges of u):

Case S1: Let v be the vertex from which u has been coordinized and let i be the color of (u,v). Let (u,w) be some down-edge of u such that $c(w)$ and $c(u)$ differ in more than 1 component. Consider the case $(u,w) \sim (u,v)$ first. Assume that there is some down-edge (u,u') with $(u,u') \not\sim (u,v)$. Then, by Lemma 2, there are edges (v,v') and (w,w') equivalent to (u,u') and edges (u',v') and (u',w') equivalent to (u,v). Let $j \neq i$ be the color of (v,v'). By ignoring all but the i-th and the j-th components of the vertices we may write $v = \binom{a}{b}, u = \binom{a}{c}, v' = \binom{d}{b}, u' = \binom{d}{c}, w' = \binom{d}{e}$. Moreover, w must be of the form $w = \binom{*}{e}$. If $* = a$ then $c(w)$ and $c(u)$ must differ in the k-th component for some $k \neq i,j$ by assumption on $c(w)$. Since w,w',u',v' and v must have the same k-th component we deduce that $c(v)$ and $c(u)$ differ in the k-th and in the i-th component. This contradiction implies $(u,u') \sim (u,v)$. If, on the other hand, $* = a$ then there must be a vertex $x = \binom{a}{e}$ and edges $(x,w') \sim (u,u')$ and $(x,u) \sim (w',u')$. By Lemma 3, $(x,w') \sim (u,w)$, which further implies $(u,u') \sim (u,v)$. This finally proves that all down-edges of u are equivalent provided $(u,w) \sim (u,v)$.
The same can be proved quite similarly in case of $(u,w) \not\sim (u,v)$ but is left to the reader. This finally implies that all down-edges of u' must be equivalent, for all down-edges (u,u') of u, and thus justifies the REDUCE-step.

Case S2.2: In (a) and (b), the existence of certain squares containing down-edges of u is explicitely tested. By similar arguments as in Case S1, the

correctness of the REDUCE-Step can be shown in case of their non-existence.

SQUARE TEST (for up- or cross-edges of v):
Similar ideas as for down-edges might be used. Again, the proofs are omitted here.

This completes the proof of Lemma 4. □

Lemma 5: If $e \sim e'$ then e and e' have the same color after termination of Algorithm FACTOR.

Proof: Let G_i be the graph induced by all vertices v with $c_i(v) \geq 1$ and $c_j(v) = 1$ for all $j \neq i, 1 \leq i, j \leq t$. It suffices to prove $G_1 \square \ldots \square G_t = G$. Assume $t \geq 2$ since nothing is to prove otherwise. By Case S1 of the SQUARE TEST we have: For all edges $(u, v) \in E$, $c(u)$ and $c(v)$ differ in exactly one component. The control flow of Algorithm FACTOR also outrules identical coordinate vectors of distinct vertices. We are left with the proof of the claim below:

An edge (v, u) is in G, $c_i(v) = a$, $c_i(u) = b$, $c_j(v) = c_j(u)$ for all $j \neq i, 1 \leq j \leq t$, if and only if the edge (v', u') is in G, $c_i(v') = a$, $c_i(u') = b$, $c_j(v') = c_j(u') = 1$ for all $j \neq i, 1 \leq j \leq t$.

The proof is by induction on the vertices saturated in BFS order. Clearly, the claim is true for all colored edges after the saturation of v_0. Now, let the claim be true for the first $k-1$ saturated vertices and let v be the vertex saturated last. All cross- and up-edges of v, and all down-edges of the vertices u such that (v, u) is an up-edge of v just have been colored. Again, we only prove the claim for down-edges of u and leave the rest to the reader. If $c(u)$ has only one component differing from 1 then the claim trivially holds. Otherwise, Case S2.2 of the SQUARE TEST is passed successfully if the following holds. Let (u, w) be a down-edge of u.

There is a down-edge (u, x) of u, $c_r(u) = a$, $c_r(x) = b \neq a$, $c_j(u) = c_j(x)$ for all $j \neq r$ if and only if there is a down-edge (w, y) of w, $c_r(w) = a$, $c_r(y) = b$, $c_j(w) = c_j(y)$ for all $j \neq r$.

All the down-edges of w have been colored while the first $k-1$ vertices have been saturated. So, by the induction hypothesis, (w, y) is in G if and only if the down-edge (w', y') of w' is in G, $c_r(w') = a$, $c_r(y') = b$, $c_j(w') = c_j(y') = 1$ for all $j \neq r$. Thus, the claim is true in this situation. If Case S2.2. is not passed successfully then a REDUCE-step follows in case of which the claim is trivially true. □

4 Complexity analysis.

We now give an analysis of the runtime and the storage required by Algorithm FACTOR. First, a word on the data structure that stores the input graph $G(V, E)$. It is assumed that G is available in adjacency list representation, see e.g., [4]. However, before starting the algorithm, we structure the adjacency list of each vertex v by splitting it into three sublists of vertices u according to whether (v, u) is an up-, cross-, or down-edge of v. During the execution of the algorithm, the sublists of v are further split into color lists when the v-edges are colored in the process of coordinizing and saturating v. The headers of the v-color lists are kept in an array of length $t = \rho(v_0)$, and vertices within each color list are linked in arbitrary order. Clearly, this representation of G occupies $O(m)$ storage and supports the access to all up- (cross-, down-) edges of v with a given color in time proportional to their number.

The coordinate vector $c(v)$, for each $v \in V$, is stored in an array of length t. In addition, entries differing from 1 are linked in increasing index order. Thus $c_i(v)$ can be accessed in constant time given $i, 1 \leq i \leq t$, and the l components differing form 1 are available in increasing order in $O(l)$ time. The storage overhead for the coordinate vectors thus is $O(nt)$ which is $O(m)$ if a vertex of minimum degree is chosen for v_0. (At this point, note that the correctness of algorithm FACTOR is independent of the choice of the start vertex.)

We are now ready to analyse the running time of FACTOR. From the overall structure of FACTOR it is evident that the following actions determine the time complexity.

(1) COMPONENT DIFFERENCE:
Given vertices u, v, determine the indices of the components in which $c(u)$ differs from $c(v)$.

(2) EDGE TEST:
Testing conditions (a) and (b) in Case S2.2 of the SQUARE TEST, for a given vertex u.

(3) REDUCE:
Given vertices $v_1, \ldots, v_k$, perform REDUCE$(v_1, \ldots, v_k)$.

(4) DOWN TEST:

Given vertices v, w such that $c(v)$ and $c(w)$ differ in exactly 2 components (the i-th and j-th, say), test the existence of down-edges (v, x), (w, x) of v and w, respectively, such that $c(x)$ differs from $c(v)$ in the i-th, and from $c(w)$ in the j-th component (or vice versa).

(5) EQUALITY TEST:
Given a vertex v, an index j, and a vector c differing from $c(v)$ exactly in the j-th component, test whether there is an up-edge (v, y) of v such that $c(y) = c$.

ad COMPONENT DIFFERENCE.
Since in $c(u)$ and $c(v)$ the components differing from 1 are linked in increasing order, $O(l)$ time clearly suffices for reporting all i with $c_i(u) \neq c_i(v)$. Here l denotes the length of the longer list.

We next show that $l \leq \log n$. Assuming the contrary implies the existence of a vertex x where $c_i(x) \geq 2$ for $k \geq 1 + \log n$ indices i. Now observe that, by the way of how components are calculated, for each combination of one's and two's of length k there is some coordinized vertex holding this combination in k of its components. So at least $2^k \geq 2n$ vertices already have been coordinized – a contradiction.

Coordinate differences are calculated at most $\rho(v)$ times when vertex v is saturated (in the SQUARE TESTs), and at most $2\rho(u)$ times when a vertex u is coordinized (in Case 2.2 and the SQUARE TEST). Since each vertex is coordinized and saturated only once, at most $3 \sum_{v \in V} \rho(v) = O(m)$ coordinate differences are calculated in total. We conclude that $O(m \log n)$ time suffices for this task.

ad EDGE TEST.
We detail this test for the case where the SQUARE TEST is applied to the set of down-edges of u: Let (u, w) and (u, w') be down-edges of u, $c_i(w) \neq c_i(u)$, $c_j(w') \neq c_j(u)$, $i \neq j$. Denote by $L_k(x)$ the color list for down-edges of x colored by k, for $1 \leq k \leq t$ and $x = u, w, w'$.

Condition (a) is tested as follows. Repeat for all $k \neq i$: If the lengths of $L_k(u)$ and $L_k(w)$ are different then (a) is violated. Otherwise, compute the sets $\{c_k(v) \mid v \text{ in } L_k(u)\}$ and $\{c_k(v) \mid v \text{ in } L_k(w)\}$. Check the identity of the sets by sorting them. (a) is violated in case of non-identity.

Condition (b) is tested similarly, but with w replaced by w', and only for $k = i$. Since computing and sorting the sets takes time proportional to their

size times a log-factor, the EDGE-TEST for u costs $O(\rho(u)\log n)$ time. The test is applied to the down-edges of a vertex u when u is coordinized, and to the cross- and up-edges of a vertex v when v is saturated. So the total time spent in EDGE TESTs amounts to $O(m\log n)$, as before.

ad REDUCE.

The execution of REDUCE $(v_1,\ldots,v_k)$ is structured as below. For $j = 1,\ldots,k$, compute the set $I_j = \{i \mid c_i(v_j) \neq 1\}$. Compute $I = \bigcup_{j=1}^{k} I_j$ by hashing into an array of length t. If $|I| = 1$ then nothing is to do. Otherwise, choose some $i \in I$ and perform IDENTIFY (i,j) for all $j \in I - \{i\}$.

IDENTIFY (i,j) replaces, for all coordinized vertices u, $c_i(u)$ by $f(c_i(u), c_j(u))$, where $f : \mathbf{N}\times\mathbf{N} \to \mathbf{N}$ is calculated as below. Sort the pairs $(c_i(u), c_j(u))$, u coordinized, using bucket sort (see [8]). For each such pair (a,b), $f(a,b) = k$ if (a,b) is (one of) the k-largest pair(s). In order to keep the color lists consistent to the new coordinate vectors, $L_i(u)$ and $L_j(u)$ are linked to a new list for color i (same for color lists of up- and cross-edges).

Since the computation of set I_j costs $O(\log n)$ time for each j, I is calculated in $O(k\log n + t)$ time. Each IDENTIFY needs $O(n)$ time for replacing components as well as for linking color lists. Note, however, that the current number of components is decreased by one. Note further that REDUCE $(v_1,\ldots,v_k)$ is called at most $3n$ times: twice for each vertex u just being saturated (in the SQUARE TESTs for cross- and up-edges) and once for each vertex u just being coordinized (in Step 2). Thereby we always have $k \leq \rho(u)$. These observations imply a total time complexity of $O(m\log n + nt) + O(nt) = O(m\log n)$ for all REDUCE-steps.

ad DOWN TEST.

For deciding the existence of the down-edges (v,x) and (w,x) we need to test whether there is a vertex x that occurs in both of $L_i(v)$ and $L_j(w)$ (or for i,j reversed). Since, in the affirmative case, $c_j(x) = c_j(v)$ and $c_i(x) = c_i(w)$, it suffices to check $c_i(w) \in \{c_i(u) \mid u \text{ in } L_i(w)\}$ and $c_j(v) \in \{c_j(u) \mid u \text{ in } L_j(w)\}$.

Each time the DOWN TEST is applied a vertex u is coordinized afterwards. If $L_i(u), L_i(v)$ and $L_j(u), L_j(w)$ have the same length, respectively, then the DOWN TEST takes $O(\rho(u))$ time. Otherwise, the DOWN TEST potentially takes $O(n)$ time, but an IDENTIFY-step will follow since u will not pass the SQUARE TEST for down-edges successfully. So the costs of all DOWN TESTs summarizes to $O(m + tn) = O(m)$.

ad EQUALITY TEST.

Let (v,y) be an up-edge of v. If y is coordinized then $c(y)$ and $c(v)$ differ

in exactly one component since y has passed the SQUARE TEST for down-edges. It thus suffices to test whether the j-th component, b, of c equals $c_j(y)$ for some up-edge (v,y). That is, $b \in \{c_j(y) \mid y \text{ in } L'_j(v)\}$ is tested, for $L'_j(v)$ denoting the (possibly up to now not complete) color list for up-edges of v with color j. By similar arguments as in the DOWN TEST, $O(m)$ time is required for all EQUALITY TESTs.

We are now ready to conclude the main theorem of this paper.

Theorem: Let G be a connected graph with n vertices and m edges. Algorithm FACTOR computes the prime factors of G with respect to Cartesian multiplication in $O(m \log n)$ time and $O(m)$ space.

References

[1] Aurenhammer, F. and Hagauer, J. Computing equivalence classes among the edges of a graph with applications. Proc. Int. Conf. Algebraic Graph Theory, 1989; Discrete Math. (to appear).

[2] Feigenbaum, J. Product graphs: some algorithmic and combinatorial properties. Ph.D.Thesis, Dept. Comput. Sci., Stanford Univ., CA, 1986.

[3] Feigenbaum, J., Hershberger, J. and Schäffer, A.A. A polynomial time algorithm for finding the prime factors of Cartesian-product graphs. Discrete Appl. Math. 12 (1985), 123 - 138.

[4] Gibbons, A. Algorithmic Graph Theory. Cambridge University Press, 1985.

[5] Graham, R.L. and Winkler, P.M. On isometric embeddings of graphs. Trans. AMS 288 (1985), 527 - 536.

[6] Hochstrasser, B. A note on Winkler's algorithm for factoring a connected graph. Proc. Int. Conf. Algebraic Graph Theory, 1989; Discrete Math. (to appear).

[7] Sabidussi, G. Graph multiplication. Math. Z. 72 (1960), 446 - 457.

[8] Sedgewick, R. Algorithms. Addison-Wesley, Reading, MA, 1983.

[9] Vizing, V.G. The Cartesian product of graphs (russ.). Engl. transl.: Comp. El. Syst. 2 (1966), 352 - 365.

[10] Winkler, P.M. Factoring a graph in polynomial time. Europ. J. Combin. 8 (1987), 209 - 212.

FINDING OUT WHETHER A VALID INEQUALITY IS FACET DEFINING

by Egon Balas, Carnegie Mellon University

We discuss a method for determining whether a valid inequality for a 0-1 polytope is facet defining. The method is based on a new procedure for generating a sequence of 0-1 points on a face of the polytope, guaranteed to be linearly independent. The sequence moves along k-dimensional subfaces of the face, and whenever this is possible for small k, the procedure becomes particularly simple. As a result, we give a new sufficient condition for a valid inequality to be facet defining, that generalizes several earlier conditions.

1. Introduction

We consider *0-1 polytopes*, i.e. bounded polyhedra of the form

$$P := \text{conv } \{x \in \{0,1\}^n : Ax \leq b\},$$

where A is an arbitrary real m×n matrix and b is an arbitrary real nonzero m-vector.

We denote $M := \{1,\ldots,m\}$, $N := \{1,\ldots,n\}$. The i-th row and j-th column of A are a^i and a_j, respectively. The *dimension* of P, denoted *dim P*, is one less than the maximum number of affinely independent points $x \in P$. The *equality set* of P is the set of inequalities $a^i x \leq b_i$ satisfied with equality by all $x \in P$; its *rank* is the maximum number of its linearly independent members. If this rank is r, then dim P = n-r.

An inequality $\alpha x \leq \alpha_o$, with $\alpha_o \neq 0$, is *valid* for P if it is satisfied by every $x \in P$. The *face* of P defined by $\alpha x \leq \alpha_o$ is $F := \{x \in P : \alpha x = \alpha_o\}$. F is proper if $\emptyset \neq F \neq P$, improper otherwise. A *facet* of P is a maximum-dimensional proper face; i.e. the face F is a facet if and only if dim F = dim P - 1.

Given a polytope P and an inequality $\alpha x \leq \alpha_o$ satisfied by all $x \in P$ and such that $\alpha x = \alpha_o$ for some (but not all) $x \in P$, it is of interest to know whether the face F defined by the inequality is a facet. There are two well-known methods for establishing this. The first one, sometimes called the direct method, consists of exhibiting dim P affinely independent points $x \in F$, which proves that F is a facet; or showing that no such set exists, which proves the opposite. The second, or indirect, method consists in showing that any inequality $\beta x \leq \beta_o$ satisfied by all $x \in P$ and such that $\beta x = \beta_o$ whenever $\alpha x = \alpha_o$, is a linear combination of $\alpha x \leq \alpha_o$ (with positive weight) and the members of the equality set of P, in which case F is a facet; or exhibiting some inequality with this property that is not such a combination, in which case F is not a facet. For a discussion of these proof methods see any of Bachem and Grötschel (1982), Pulleyblank (1983), Schrijver (1986) or Nemhauser and Wolsey (1988). The difficulty with the direct method does not consist in finding dim P points in F, which is typically easy, but in finding dim P affinely independent points or showing that no such set of points exists, which is

---

The research underlying this report was supported by Grant ECS-8601660 of the National Science Foundation and Contract N00014-85-K-0198 with the Office of Naval Research.

often hard.

In this paper we discuss a variant of the direct method, based on a procedure for generating a sequence of 0-1 points in F that are guaranteed to be linearly independent. The procedure comes in two versions. The first one, called 2-reduction, is straightforward but may stop short of generating the required number of points. Nevertheless, it yields a sufficient condition for an inequality to be facet defining, that generalizes some well known sufficient conditions from the literature and makes it easy to prove the facet inducing property of large classes of inequalities. The second, more general version, called r-reduction, is always applicable, but computationally more cumbersome. It provides a weaker sufficient condition, as well as a necessary and sufficient condition for an inequality to be facet inducing.

The problem examined here was addressed earlier in a more general context by Edmonds, Lovász and Pulleyblank (1982), who gave an algorithm for determining the dimension of an arbitrary polyhedron P*, that is polynomial if the problem of maximizing a linear function over P* is polynomially solvable. The algorithm constructs dim P*+1 affinely independent points in P* and n-dim P* affinely independent equations satisfied by every point in P*. By contrast, our procedure is restricted to 0-1 polytopes and does not enable its user to identify the equality set of the face F that it examines. Its focus in trying to find linearly independent points in F is to move along k-dimensional subfaces for k = 1,2, etc. Whenever this is possible for a small k, the procedure is particularly simple and therein lies its main advantage.

Our paper is organized as follows. The next two sections discuss 2-reduction and r-reduction, respectively; section 4 takes up the case of positive 0-1 polytopes; sections 5 and 6 discuss application to vertex packing and set covering, respectively.

2. 2-Reduction

Let $\alpha x \leq \alpha_o$ be a valid inequality for P, with $\alpha_o \neq 0$, $\alpha_j \neq 0$, $j \in N$, and let $F := \{x \in P : \alpha x = \alpha_o\}$. We will consider certain partitions of the index set N. A *partition* $\pi := \{N_1, \ldots, N_s\}$ of N is a collection of disjoint, nonempty subsets N_k of N whose union is N. If the partition has s members (subsets), we call it an s-partition. The *trivial* partition is the n-partition $\pi = \{\{1\}, \ldots, \{n\}\}$. The *improper* partition is the 1-partition $\pi = \{N\}$.

With any partition $\pi = \{N_1, \ldots, N_s\}$ and any $x \in F$ we associate a vector $\pi(x) \in \mathbb{R}^s$, $\pi(x) = (\pi_1(x), \ldots, \pi_s(x))$, defined by

$$\pi_k(x) = \Sigma(\alpha_j x_j : j \in N_k) \qquad k = 1, \ldots, s, \tag{1}$$

and called the *π-pattern* of x with respect to $\alpha x \leq \alpha_o$. We will say that the partition $\pi = \{N_1, \ldots, N_s\}$ of N is *2-reducible* with respect to $\alpha x \leq \alpha_o$ if there exists a pair $i, j \in \{1, \ldots, s\}$, and a pair $x, y \in F$, such that

$$\pi_k(x) \begin{cases} \neq \pi_k(y) & k = i, j \\ = \pi_k(y) & k \in \{1, \ldots, s\} \setminus \{i, j\}. \end{cases} \tag{2}$$

Note that when π is the trivial partition of N, then (2) becomes

$$\alpha_k x_k \begin{cases} \neq \alpha_k y_k & k = i, j \\ = \alpha_k y_k & k \in \{1, \ldots, s\} \setminus \{i, j\} \end{cases} \tag{3}$$

If π is 2-reducible and i,j is the pair for which relation (2) holds, we call *2-reduction* the operation that replaces π with

$$\pi' := (\pi \backslash \{N_i, N_j\}) \cup N_{ij}, \tag{4}$$

here $N_{ij} := N_i \cup N_j$.

Note that while $\pi(x) \neq \pi(y)$, as a result of the 2-reduction we obtain $\pi'_k(x) = \pi'_k(y)$ for all subset indices k pertaining to the partition π'.

Consider now the following

2-Reduction Procedure (with respect to $\alpha x \leq \alpha_o$).

Initialization. Let π^1 be the trivial partition of N, let $x^1 \varepsilon F$, and t := 1.

Iterative Step. Given $\pi^t := \{N_1, \ldots, N_s\}$ and $x^t \varepsilon F$, find $x^{t+1} \varepsilon F$ such that

$$\pi_k^t(x^{t+1}) \begin{cases} \neq \pi_k^t(x^t) & k = i,j \\ = \pi_k^t(x^t) & k \varepsilon \{1,\ldots,s\} \backslash \{i,j\} \end{cases} \tag{5}$$

for some pair $i,j \varepsilon \{1,\ldots,s\}$. If no such x^{t+1} exists, stop. Otherwise let $N_{ij} := N_i \cup N_j$,

$$\pi^{t+1} := (\pi^t \backslash \{N_i, N_j\}) \cup N_{ij}, \tag{6}$$

t := t+1, and repeat.

Lemma 1. If $\pi^p = \{N_1, \ldots, N_s\}$ is the current partition of N at some iteration p of the 2-reduction procedure, then the π^p-patterns of all the vectors $x^t \varepsilon F$, t = 1,...,p, used in the procedure up to iteration p, are the same; i.e.

$$\pi^p(x^t) = (C_1, \ldots, C_s), \qquad t = 1, \ldots, p,$$

where $C_1, \ldots, C_s$ are constants independent of t.

Proof. We use induction on p. For p = 1 the statement is trivially true. Suppose it is true for p = 1,...,q, and let p = q+1 ≥ 2.. If $\pi^q = \{N_1, \ldots, N_s\}$, by hypothesis we have

$$\pi_k^q(x^t) = C_k \text{ for } t = 1, \ldots q$$

for all $k \varepsilon \{1,\ldots,s\}$. At the (q+1)-st application of the 2-reduction step, we have for some $i,j \varepsilon \{1,\ldots,s\}$

$$\pi_k^q(x^{q+1}) = C_k \text{ for } k \varepsilon \{1,\ldots,s\} \backslash \{i,j\}$$

and

$$\begin{aligned} \pi_i^q(x^{q+1}) + \pi_j^q(x^{q+1}) &= \alpha_o - \Sigma(\pi_k^q(x^{q+1}) : k \varepsilon \{1,\ldots,s\} \backslash \{i,j\}) \\ &= \alpha_o - \Sigma(C_k : k \varepsilon \{1,\ldots,s\} \backslash \{i,j\}) \\ &= C_i + C_j \end{aligned}$$

Hence for the partition $\pi^{q+1} = \{N'_1, \ldots, N'_{s-1}\}$, where the subsets $N'_1, \ldots, N'_{s-1}$ come from renumbering the subsets N_k, $k \varepsilon \{1,\ldots,s\} \backslash \{i,j\}$ and $N_{ij} = N_i \cup N_j$, we have

$$\pi_k^{q+1}(x^t) = C_k \text{ for } t = 1, \ldots, q+1 \text{ and } k = 1, \ldots, s-1,$$

with $C_{k_*} = C_i + C_j$ for the new subset N'_{k_*} corresponding to N_{ij}. This completes the induction. ■

Theorem 2. If the 2-Reduction Procedure stops at iteration p, then π^p is an s-partition with s = n-p+1, and the p vectors $x^t \varepsilon F$, t = 1,...,p, used in the procedure, are linearly independent.

Proof. Since π^1 is an n-partition and at every iteration the number of

subsets in the partition decreases by one, π^p is an s-partition with $s = n-p+1$. To prove that the p vectors $x^t \varepsilon F$, $t = 1,\ldots,p$, are linearly independent, again we use induction on p. For $p = 1$ the statement is trivially true. Suppose it is true for $p = 1,\ldots,q$, and let $\pi^q = \{N_1,\ldots,N_s\}$, $p = q+1 \geq 2$. By the induction hypothesis, the vectors $x^t \varepsilon F$, $t = 1,\ldots,q$, are linearly independent. Suppose now, for the sake of contradiction, that the vectors $x^1,\ldots,x^{q+1} \varepsilon F$ are not. Then there exist scalars λ_t, $t = 1,\ldots,q$, such that $x^{q+1} = \Sigma(x^t\lambda_t : t = 1,\ldots,q)$; and hence for $k = 1,\ldots,s$, we have (by substitution)

$$\pi_k^q(x^{q+1}) = \Sigma(\pi_k^q(x^t)\lambda_t : t = 1,\ldots,q),$$

which from Lemma 1 can be written as

$$\pi_k^q(x^{q+1}) = C_k\Sigma(\lambda_t : t = 1,\ldots,q), \qquad k = 1,\ldots,s.$$

On the other hand, from (5) and Lemma 1 we have

$$\pi_k^q(x^{q+1}) \begin{cases} \neq C_k & k = i,j \\ = C_k & k\varepsilon\{1,\ldots,s\}\backslash\{i,j\}, \end{cases}$$

and so we obtain on the one hand $\Sigma(\lambda_t : t = 1,\ldots,q) \neq 1$ and on the other $\Sigma(\lambda_t : t = 1,\ldots,q) = 1$, a contradiction. This completes the induction. ■

The following Corollaries are immediate consequences of Theorem 2.

Corollary 3. $\dim F \geq n-s$ $(= p-1)$.

Corollary 4. If the 2-Reduction Procedure stops with the improper partition $\pi^n = \{N\}$, then F is a facet of P.

Next we address the question of the complexity of 2-reduction.

Proposition 5. Given a partition $\pi = \{N_1,\ldots,N_s\}$ of N, a pair $i,j\varepsilon\{1,\ldots,s\}$, and a vector $x^t \varepsilon F$, let $f(n)$ be the complexity function of finding $x^{t+1} \varepsilon F$ that satisfies (5) or showing that no such x^{t+1} exists. Then the complexity of the 2-reduction procedure is $O(n^3 f(n))$.

Proof. At each iteration, the problem stated in the theorem has to be solved at most $O(n^2)$ times, once for each pair $i,j\varepsilon\{1,\ldots,s\}$. Since these are at most n iterations, the statement follows. ■

Example 1. Let P be the convex hull of those $x\varepsilon\{0,1\}^9$ satisfying

$$5x_1 + 3x_2 + 2x_3 + x_4 + 3x_5 + x_6 + x_7 + x_8 + x_9 \leq 8$$

$$4x_1 + 6x_2 + 4x_3 + 5x_4 + 3x_5 + 2x_6 + 2x_7 + 2x_8 + 2x_9 \leq 12,$$

and consider the inequality

$$3x_1 + 3x_2 + 2x_3 + 2x_4 + 2x_5 + x_6 + x_7 + x_8 + x_9 \leq 6,$$

obtained by adding the two inequalities that define P, dividing the resulting inequality by 3, and rounding down the right side to the nearest integer. This is a special case of Chvátal's (1973) procedure, and the resulting inequality is obviously valid for P. We want to know whether it is facet defining.

We start with the trivial partition $\pi^1 := \{\{1\},\ldots,\{9\}\}$ and with $x^1 = (1,0,1,0,0,1,0,0,0)$. At the first iteration we use $x^2 = (1,0,1,0,0,0,1,0,0)$ to obtain $\pi^2 := \{\{1\},\ldots,\{5\},\{6,7\},\{8\},\{9\}\}$; etc. The sequence of vectors $x^t \varepsilon F$ and partitions π^t of N is listed below:

$x^1 = (1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0)$ $\qquad$ $\pi^1 = \{\{1\},\ldots,\{9\}\}$

$x^2 = (1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0)$ $\qquad$ $\pi^2 = \{\{1\},\ldots,\{5\},\{6,7\},\{8\},\{9\}\}$

$x^3 = (1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0)$ $\quad \pi^3 = \{\{1\},\ldots,\{5\},\{6,7,8\},\{9\}\}$
$x^4 = (1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1)$ $\quad \pi^4 = \{\{1\},\ldots,\{5\},\ \{6,7,8,9\}\}$
$x^5 = (1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1)$ $\quad \pi^5 = \{\{1\},\{2\},\{3,4\},\{5\},\{6,\ldots,9\}\}$
$x^6 = (1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1)$ $\quad \pi^6 = \{\{1\},\{2\},\{3,4,5\},\{6,\ldots,9\}\}$
$x^7 = (0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1)$ $\quad \pi^7 = \{\{1,2\},\{3,4,5\},\{6,\ldots,9\}\}$
$x^8 = (0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1)$ $\quad \pi^8 = \{\{1,2\},\{3,\ldots,9\}\}$
$x^9 = (1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$ $\quad \pi^9 = \{1,\ldots,9\}.$

Since π^9 is the improper partition, the inequality under examination defines a facet of P.■

3. r-Reduction

The 2-reduction procedure can be generalized to an r-reduction procedure that starts with 2-reduction and continues beyond the point where the latter fails. While the 2-reduction procedure provides a sufficient condition for F to be a facet of P, this more general reduction procedure provides a necessary and sufficient condition.

Given a partition $\pi = \{N_1,\ldots,N_s\}$ of N, with $\{1,\ldots,s\} =: S$, we will say that π is *r-reducible* with respect to $\alpha x \le \alpha_o$ if there exists $R \subseteq S$, $|R| \ge 2$, and a collection of points $x^t \in F$, $t = 1,\ldots,r = |R|$, whose π-patterns $\pi(x^1),\ldots,\pi(x^r)$ are linearly independent and satisfy

$$\pi_k(x^t) = C_k, \qquad t = 1,\ldots,r;\ k\in S\backslash R. \tag{7}$$

If π is r-reducible and R is the subset of S for which (7) holds, we call *r-reduction* the operation that replaces π with

$$\pi' := (\pi\backslash\{N_k : k\in R\}) \cup N_R, \tag{8}$$

where R is the subset of S used in (7) and $N_R := \cup(N_k : k\in R)$. Clearly, r-reduction generalizes the operation of 2-reduction introduced in the previous section.

As in the case of 2-reduction, it is worth noting that the partition π' obtained by (8) satisfies $\pi'_k(x^1) = \ldots = \pi'_k(x^r)$ for all subset indices k pertaining to the partition π'.

We will call a partition of N *valid* with respect to $\alpha x \le \alpha_o$ if it was obtained from the trivial partition by a sequence of r-reduction steps (where r may differ from one step to another).

Consider now the following

Reduction Procedure (with respect to $\alpha x \le \alpha_o$).

Initialization. Let π^1 be the trivial partition of N, let $x^1 \in F$ and $t := 1$.

Iterative Step. Given $\pi^t = \{N_1,\ldots,N_s\}$ and $x^t \in F$, find r-1 vectors $x^i \in F$, $i = t+1,\ldots,t+r-1$, $2 \le r \le s$, such that the r vectors $\pi^t(x^i)$, $i = t, t+1,\ldots, t+r-1$ are linearly independent and satisfy

$$\pi^t_k(x^{t+h}) = \pi^t_k(x^t), \qquad h = 1,\ldots,r-1;\ k\in S\backslash R \tag{9}$$

for some $R \subseteq S := \{1,\ldots,s\}$, $|R| = r$.

If no such set of r-1 vectors $x^i \in F$ exists, stop; otherwise let $N_R := \cup(N_k : k\in R)$,

$$\pi^{t+1} := (\pi^t\backslash\{N_k : k\in R\}) \cup N_R, \tag{10}$$

$t := t+1$ and repeat.

We will use the following analog of Lemma 1.

Lemma 6. If $\pi^p = \{N_1,\ldots,N_s\}$ is the current partition of N at some itera-

tion p of the reduction procedure, and $x^t \varepsilon F$, $t = 1,\ldots,v$ are the vectors used to obtain π^p, then the π^p-patterns of all x^t, $t = 1,\ldots,v$ are the same, i.e.

$$\pi^p(x^t) = (C_1,\ldots,C_s), \qquad t = 1,\ldots,v,$$

where $C_1,\ldots,C_s$ are constants independent of t.

Proof. Again we use induction on p. For $p = 1$ the statement is trivially true. Suppose it is true for $p = 1,\ldots,q$, and let $p = q+1 \geq 2$. Further, let $\pi^q = \{N_1,\ldots,N_s\}$ with $\{1,\ldots,s\} =: S$. By the induction hypothesis, $\pi^q(x^i) = C_k$, $k\varepsilon S$, $i = 1,\ldots,v$. At the iteration which yields π^{q+1}, we have for some $R \subseteq S$, $|R| = r_q$, and for $i = 1,\ldots,r_q-1$, $\pi^q_k(x^{v+1}) = C_k$ for $k\varepsilon S\backslash R$ and

$$\begin{aligned}\Sigma(\pi^q_k(x^{v+i}):k\varepsilon R) &= \alpha_o - \Sigma(\pi^q_k(x^{v+1}):k\varepsilon S\backslash R)\\ &= \alpha_o - \Sigma(C_k:k\varepsilon S\backslash R)\\ &= \Sigma(C_k:k\varepsilon R).\end{aligned}$$

The resulting partition is then of the form $\pi^{q+1} = \{N'_1,\ldots,N'_{s-r_q+1}\}$, where the subsets $N'_1,\ldots,N'_{s-r_q+1}$ come from the renumbering of N_k, $k\varepsilon S\backslash R$, and $N_R = \cup\,(N_k:k\varepsilon R)$. For this partition we then have $\pi^{q+1}_k(x^i) = C_k$ for $k = 1,\ldots,s-r_q+1$ and $i = 1,\ldots,v+r_q-1$, with $C_{k_*} = \Sigma(C_\ell:\ell\varepsilon R)$ for the index k_* corresponding to the new subset N_R. This completes the induction. ■

We now prove for the general reduction procedure a property analogous to the one proved in Theorem 2 for the 2-reduction procedure.

Theorem 7. Suppose the Reduction Procedure stops with the partition π^p after performing a sequence of p-1 r-reductions, with $r = r_1,\ldots,r_{p-1}$, respectively, and let $\rho := 1 + (r_1-1) + \ldots + (r_{p-1}-1)$. Then π^p is an s-partition with $s = n-\rho+1$; and the ρ vectors $x^t\varepsilon F$, $t = 1,\ldots,\rho = n-s+1$, used in the procedure, are linearly independent.

Proof. Since the improper partition has n subsets and each r-reduction reduces the number of subsets by r-1, we have $s = n - ((r_1-1) + \ldots + (r_{p-1}-1)) = n - \rho + 1$, as claimed. Suppose now that the vectors $x^t\varepsilon F$ used in the procedure are linearly dependent. Then there exist scalars λ_t, $t = 1,\ldots,\rho$, not all zero, such that $\Sigma(x^t\lambda_t: t = 1,\ldots,\rho) = 0$. Let $\pi^{p-1} = \{N_1,\ldots,N_m\}$ be the partition of N from which π^p was obtained as the result of an r_{p-1}-reduction. Then for $k = 1,\ldots,m$, $\Sigma(\pi^{p-1}_k(x^t)\lambda_t: t = 1,\ldots,\rho) = 0$. The $r_{p-1}-1$ vectors used in the last reduction step are $x^t\varepsilon F$ for $t = \rho - r_{p-1}+2, \ldots, \rho$ and from the last equation we have for $k = 1,\ldots,m$,

$$\begin{aligned}0 &= \Sigma(\pi^{p-1}_k(x^t)\lambda_t: t= 1,\ldots,\rho-r_{p-1}+1) + \Sigma(\pi^{p-1}_k(x^t)\lambda_t: t = \rho-r_{p-1}+2,\ldots,\rho) =\\ &= \pi^{p-1}_k(x^{\rho-r_{p-1}+1})\mu_1 + \Sigma(\pi^{p-1}_k(x^{\rho-r_{p-1}+j})\mu_j: j = 2,\ldots,r_{p-1})\end{aligned} \tag{11}$$

where $\mu_1 := \Sigma(\lambda_t: t = 1,\ldots,\rho-r_{p-1}+1)$, $\mu_j := \lambda_{p-r_{p-1}+j}$ for $j = 2,\ldots,r_{p-1}$. Here the second equation follows from the fact that, according to Lemma 6,

$$\pi^{p-1}_k(x^t) = C_k \text{ for } t = 1,\ldots,\rho-r_{p-1}+1 \text{ and } k = 1,\ldots,m.$$

But (11) contradicts the linear independence of the π^{p-1}-patterns of the r_{p-1} vectors $x^t\varepsilon F$, $t = \rho-r_{p-1}+1,\ldots,\rho$ used in the last reduction step; which proves statement. ■

As in the case of the 2-reduction procedure, the following corollary is an immediate consequence of Theorem 7. Let p, s and ρ be as in Theorem 7.

Corollary 8. $\dim F \geq n-s\ (= \rho-1)$.

We now state a necessary and sufficient condition for F to be a facet of P.

Theorem 9. Let π be any valid s-partition of N with respect to $\alpha x \leq \alpha_o$. Then F is a facet of P if and only if there exist s + dim P - n vectors $x \in F$ whose π-patterns are linearly independent.

Proof. Let $x^t \in F$, $t = 1,\ldots,n-s+1$, be the vectors used to obtain π from the trivial partition. Recall that the π-patterns of all vectors x^t are the same (Lemma 6), say $\pi(x^t) = (C_1,\ldots,C_s)$, $t = 1,\ldots,n-s+1$. We claim that $s \geq n-\dim P$. This is obviously true of the trivial partition (for which $s = n$), as 0 is certainly a lower bound on dim P; and every r-reduction step, while reducing the size of s by r-1, also adds r-1 new vectors $x \in F$ to the current linearly independent set, thereby increasing the lower bound on dim P by the same amount. This proves the claim.

Sufficiency. Now suppose there exist $p := s + \dim P - n$ vectors $x \in F$ whose π-patterns are linearly independent. W.l.o.g. we may assume that at least one of these vectors belongs to the set of those x^t, $t = 1,\ldots,n-s+1$, used in the reduction procedure; on the other hand, at most one of them can belong to that set, since the π-patterns of all vectors x^t, $t = 1,\ldots,n-s+1$, are equal to $(C_1,\ldots,C_s)$. So let $x^t \in F$, $t = n-s+2, n-s+3,\ldots, n-s+p$ be the p-1 vectors whose π-patterns are different from $(C_1,\ldots,C_s)$. By the same argument as in the proof of Theorem 7, we can show that the $n-s+p = \dim P$ vectors $x^t \in F$, $t = 1,\ldots,n-s+p$, are linearly independent. For suppose not. Then there exist scalars λ_t, $t = 1,\ldots,n-s+p$, not all zero, such that $\Sigma(x^t\lambda_t : t = 1,\ldots,n-s+p) = 0$ and hence

$$\Sigma(\pi_k(x^t)\lambda_t : t = 1,\ldots,n-s+p) = 0 \text{ for } k = 1,\ldots,s.$$

This last equation yields for $k = 1,\ldots,s$,

$$\begin{aligned} 0 &= \Sigma(\pi_k(x^t)\lambda_t : t = 1,\ldots,n-s+1) + \Sigma(\pi_k(x^t)\lambda_t : t = n-s+2,\ldots,n-s+p) \\ &= \pi_k(x^{n-s+1})\mu_1 + \Sigma(\pi_k(x^{n-s+j})\mu_j : j = 2,\ldots,p), \end{aligned} \tag{12}$$

where $\mu_1 := \Sigma(\lambda_t : t = 1,\ldots,n-s+1)$, $\mu_j := \lambda_{n-s+j}$ for $j = 2,\ldots,p$. However, equation (12) contradicts the linear independence of the π-patterns of the p vectors $x^t \in F$, $t = n-s+1,\ldots,n-s+p$, which proves the sufficiency of the condition in the Theorem.

Necessity. Suppose F is a facet of P. Then there exists a set T of dim P linearly independent vectors $x \in F$. Further, w.l.o.g. we may assume that the vectors x^t, $t = 1,\ldots,n-s+1$, used to obtain π from the trivial partition, belong to T. Let the remaining p-1 vectors of T be $x^t, t = n-s+2,\ldots,n-s+p$ (= dim P), and let X be the $(n-s+p)\times n$ matrix whose t-th row is x^t, $t = 1,\ldots,n-s+p$. We claim that the π-patterns of the p vectors x^t, $t = n-s+1,\ldots,n-s+p$, are linearly independent. For consider the $(n-s+p)\times s$ matrix Π whose t-th row is the π-pattern $\pi(x^t)$ of x^t for $t = 1,\ldots,n-s+p$, and suppose for the sake of contradiction that $\text{rank}(\Pi) < p$. Then for every subset V of $S := \{1,\ldots,s\}$ of cardinality $p = s + \dim P - n$, there exist scalars λ_k, $k = 1,\ldots,p$, not all zero, such that $\Sigma(\Pi_k\lambda_k : k \in V) = 0$ where Π_k is the k-th column of Π.

On the other hand, since $\text{rank}(X) = \dim P = n-s+p$, there exists a subset W of N of cardinality n-s+p, such that $\Sigma(X_j\mu_j : j \in W) = 0$ implies $\mu_j = 0$, $j \in W$. We claim that w.l.o.g. we may assume $W \cap N_k \neq \emptyset$ for at least p indices $k \in S$ (here N_k is the k-th subset of the partition π). For if not, i.e. if $W \cap N_k = \emptyset$ for more than s-p indices $k \in S$, then $|W| < n-s+p$, contrary to our assumption on $|W|$. Now let V^* be the set of those $k \in S$ such that $W \cap N_k \neq \emptyset$. Since

$|V^*| \geq p$, there exists scalars λ_k, $k\varepsilon V^*$, not all zero, such that $\Sigma(\Pi_k \lambda_k : k\varepsilon V^*) = 0$, or, using the definition of Π,

$$0 = \Sigma(\Sigma(\alpha_j X_j : j\varepsilon W \cap N_k)\lambda_k : k\varepsilon V^*)$$
$$= \Sigma(X_j \mu_j : j\varepsilon W),$$

where $\mu_j = \alpha_j \lambda_k$ for $j\varepsilon W$ and $k\varepsilon V^*$ such that $j\varepsilon W \cap N_k$, and where the second equation holds since $W \subseteq (\cup (N_k : k\varepsilon V^*)$.

As mentioned above, the last equation implies $\mu_j = 0$, $j\varepsilon W$, which in turn, together with $\alpha_j \neq 0$, $j\varepsilon N$, implies $\lambda_k = 0$, $k\varepsilon V^*$, a contradiction. This proves that the π-patterns of $p = s + \dim P - n$ vectors x^t, $t = n-s+1, \ldots, n-s+p$, are linearly independent. ■

We notice that for an arbitrary 0-1 polytope P, even if F is a facet of P, the reduction procedure may stop with an s-partition π of N such that $s > n - \dim P + 1$. This can happen if at some iteration there exists no set of r vectors $x^t \varepsilon F$ whose π-patterns are linearly independent *and* satisfy (9), for any $r\varepsilon[2, s + \dim P - n]$, but there exists a set of $s + \dim P - n$ vectors in F whose π-patterns are linearly independent *without* satisfying (9). For a full dimensional polytope, however, this cannot happen, as the next Corollary shows.

Corollary 10. Let P be full-dimensional. Then F is a facet of P if and only if the improper partition is valid with respect to $\alpha x \leq \alpha_o$.

Proof. Sufficiency follows from Corollary 8. Now suppose F is a facet of P, and let π be the s-partition with which the reduction procedure stops. If $s > 1$, from Theorem 9 there exist $s + \dim P - n = s$ vectors $x\varepsilon F$ whose π-patterns are linearly independent. But since P is full-dimensional, these vectors form a set for which condition (9) of the iterative step is trivially satisfied as it becomes vacuous (since R = S), hence a set that can be used for another iteration of the reduction procedure, contrary to the assumption that the procedure has stopped with π. Thus $s = 1$ and the improper partition is valid with respect to $\alpha x \leq \alpha_o$. ■

Example 2. Consider the same polytope P and face F as in Example 1, but start with a different $x\varepsilon F$, say $x^1 = (0, 0, 1, 1, 1, 0, 0, 0, 0)$. Then 2-reduction is not applicable to the trivial partition π^1, since there is no vector $x^2 \varepsilon F$ to satisfy condition (9) for r = 2 (with x^1 in the role of x^t). The smallest r for which (9) can be satisfied is r = 4, with x^2, x^3, x^4 and the resulting π^2 shown below:

	$x^1 = (0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0)$	$\pi^1 = \{\{1\}, \ldots, \{9\}\}$
r = 4:	$x^2 = (0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0)$	
	$x^3 = (0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0)$	
	$x^4 = (0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0)$	$\pi^2 = \{\{1\},\{2\},\{3\},\{5\},\{4,6,7,8\},\{9\}\}$

At all the remaining iterations, 2-reductions are possible:

r = 2:	$x^5 = (0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1)$	$\pi^3 = \{\{1\},\{2\},\{3\},\{5\},\{4,6,7,8,9\}\}$
r = 2:	$x^6 = (0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1)$	$\pi^4 = \{\{1\},\{2\},\{5\},\{3,4,6,7,8,9\}\}$
r = 2:	$x^7 = (0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1)$	$\pi^5 = \{\{1\},\{2\},\{3,\ldots,9\}\}$
r = 2:	$x^8 = (1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1)$	$\pi^6 = \{\{2\},\{1,\ldots,9\}\}$
r = 2:	$x^9 = (0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1)$	$\pi^7 = \{1,\ldots,9\}$. ■

4. The Case of Positive 0-1 Polytopes

In all the results so far we have assumed that the valid inequality $\alpha x \le \alpha_o$ defining F has no zero coefficients. While this is a genuine restriction for a general 0-1 polytope, it is no restriction at all for positive 0-1 polytopes, as we will presently show.

By a *positive 0-1 polytope* we mean the convex hull of 0-1 points x satisfying a system like $Ax \le b$ or $Ax \ge b$, with $A \ge 0$ and $b > 0$. Let A be mxn, and let

$$P^<(A,b) := \text{conv } \{x\varepsilon\{0,1\}^n : Ax \le b\}$$
$$P^>(A,b) := \text{conv } \{x\varepsilon\{0,1\}^n : Ax \ge b\}$$

Whenever it does not create confusion, we write $P^<$ and $P^>$ for $P^<(A,b)$ and $P^>(A,b)$, respectively.

It is not hard to see that $P^<$ is full dimensional if and only if $a_j \le b$ for all $j\varepsilon N$, whereas $P^>$ is full dimensional if and only if $a_j \le Ae-b$ for all $j\varepsilon N$. For the rest of this section we assume that $P^<$ and $P^>$ are full dimensional.

An inequality $\alpha x \le \alpha_o$, with $\alpha_o > 0$, valid for $P^<$, is *maximal* if there exists no $k\varepsilon N$ and $\alpha'_k > \alpha_k$ such that

$$\alpha'_k x_k + \Sigma(\alpha_j x_j : j\varepsilon N\backslash\{k\}) \le \alpha_o$$

is valid for $P^<$. Similarly, an inequality $\alpha x \ge \alpha_o$, with $\alpha_o > 0$, valid for $P^>$, is *minimal* if there exists no $k\varepsilon N$ and $\alpha''_k < \alpha_k$ such that

$$\alpha''_k x_k + \Sigma(\alpha_j x_j : j\varepsilon N\backslash\{k\}) \ge \alpha_o$$

is valid for $P^>$.

We denote by $F^< := \{x\varepsilon P^< : \alpha x = \alpha_o\}$ and $F^> := \{x\varepsilon P^> : \alpha x = \alpha_o\}$ the faces of $P^<$ and $P^>$ defined by $\alpha x \le \alpha_o$ and $\alpha x \ge \alpha_o$, respectively.

Proposition 11A. A valid inequality $\alpha x \le \alpha_o$ for $P^<$, with $\alpha_o > 0$, is maximal if and only if for every $j\varepsilon N$ there exists $x\varepsilon F^<$ such that $x_j = 1$.

Proposition 11B. A valid inequality $\alpha x \ge \alpha_o$ for $P^>$, with $\alpha_o > 0$, is minimal if and only if for every $j\varepsilon N$ there exists $x\varepsilon F^>$ such that $x_j = 1$.

Corollary 12A. If $\alpha x \le \alpha_o$, with $\alpha_o > 0$, is a maximal valid inequality for $P^<$, then $0 \le \alpha_j \le \alpha_o$ for all $j\varepsilon N$.

The property stated in Corollary 12A does not have its exact analog for minimal valid inequalities for $P^>$. Thus for $P^>$ defined by the system

$$x_1 + x_2 + x_3 \ge 2, \qquad x_1, x_2, x_3 = 0 \text{ or } 1,$$

the inequality

$$-x_1 + 2x_2 + 2x_3 \ge 1$$

is both valid and minimal, although $\alpha_1 = -1 < 0$ and $\alpha_2 = \alpha_3 = 2 > \alpha_o$. However, the following weaker property holds.

Corollary 12B. If $\alpha x \ge \alpha_o$, with $\alpha_o > 0$, is a minimal valid inequality for $P^>$ with $\alpha_j \ge 0$, $j\varepsilon N$, then $\alpha_j \le \alpha_o$ for all $j\varepsilon N$.

This situation suggests that we next examine the connection between $P^<(A,b)$ and the associated polytope

$$P^>(A,\tilde{b}) := \{y\varepsilon\{0,1\}^n \mid Ay \ge \tilde{b} := Ae-b\}$$

obtained by complementing the variables x_j, $j\varepsilon N$, in the definition of $P^<(A,b)$. We first note that if the inequality $\alpha x \le \alpha_o$, valid for $P^<(A,b)$, is maximal, it does not follow that the inequality $\alpha y \ge \alpha e-\alpha_o$, obviously valid for $P^>(A,\tilde{b})$, is minimal. Indeed, $2x_1 + x_2 + x_3 \le 3$ is a maximal valid inequality

for the polytope $P^<$ defined by

$$x_1 + x_2 + x_3 \le 2, \qquad x_1, x_2, x_3 = 0 \text{ or } 1,$$

but the corresponding inequality $2y_1 + y_2 + y_3 \ge 1$ is not minimal for $P^>(A,\tilde{b})$, which is defined by

$$y_1 + y_2 + y_3 \ge 1, \qquad y_1, y_2, y_3 = 0 \text{ or } 1.$$

These considerations prompt us to introduce the following stronger notion of maximality and minimality of inequalities. A valid inequality $\alpha x \le \alpha_o$ for $P^<(A,b)$ will be called *strongly maximal* if it is maximal, and the inequality $\alpha y \ge \alpha e - \alpha_o$ is minimal for $P^>(A,\tilde{b})$. Similarly, a valid inequality $\alpha x \ge \alpha_o$ for $P^>(A,b)$ will be called *strongly minimal* if it is minimal, and the inequality $\alpha y \le \alpha e - \alpha_o$ is maximal for $P^<(A,\tilde{b})$.

Proposition 13A. A valid inequality $\alpha x \le \alpha_o$ for $P^<$ is strongly maximal if and only if for every $j \in N$ there exists $x \in F^<$ such that $x_j = 1$ and $x' \in F^<$ such that $x'_j = 0$.

Proof. Follows from applying the definition and setting $x'_j = 1 - y_j$, $j \in N$. ■

Proposition 13B. A valid inequality $\alpha x \ge \alpha_o$ for $P^>$ is strongly minimal if and only if for every $j \in N$ there exists $x \in F^>$ such that $x_j = 1$ and $x' \in F^>$ such that $x'_j = 0$.

Proof. Same as for Proposition 13A. ■

Corollary 14A. If $\alpha x \le \alpha_o$ is a strongly maximal valid inequality for $P^<$, then $0 \le \alpha_j \le \min\{\alpha_o, \alpha e - \alpha_o\}$ for all $j \in N$.

Proof. If $\alpha x \le \alpha_o$ is maximal for $P^<$ then from Corollary 12A, $0 \le \alpha_j \le \alpha_o$. If, in addition, $\alpha y \ge \alpha e - \alpha_o$ is minimal for $P^>(A,\tilde{b})$, then from Corollary 12B, $\alpha_j \le \alpha e - \alpha_o$. ■

Corollary 14B. If $\alpha x \ge \alpha_o$ is a strongly minimal valid inequality for $P^>$, then $0 \le \alpha_j \le \min\{\alpha_o, \alpha e - \alpha_o\}$ for all $j \in N$.

Proof. If $\alpha y \le \alpha e - \alpha_o$ is maximal for $P^<(A,\tilde{b})$, then $0 \le \alpha_j \le \alpha e - \alpha_o$ for all $j \in N$. If $\alpha x \ge \alpha_o$ is minimal for $P^>(A,b)$ with $\alpha_j \ge 0$, $j \in N$, then $\alpha_j \le \alpha_o$ for all $j \in N$. ■

Proposition 15A. The inequality $\alpha x \le \alpha_o$ defines a facet of $P^<(A,b)$ if and only if $\alpha y \ge \alpha e - \alpha_o$ defines a facet of $P^>(A,\tilde{b})$.

Proof. Follows from the fact that $x \in P^<(A,b)$ if and only if $y \in P^>(A,\tilde{b})$ for $y = e - x$, and a collection of points $x^i \in P^<(A,b)$, $i = 1,\ldots,t$ is affinely independent if and only if the collection of points $y^i := e - x^i$, $i = 1,\ldots,t$, is affinely independent. ■

Proposition 15B. The inequality $\alpha x \ge \alpha_o$ defines a facet of $P^>(A,b)$ if and only if $\alpha y \le \alpha e - \alpha_o$ defines a facet of $P^<(A,\tilde{b})$.

Proof. Same as for Proposition 15A. ■

It follows directly from Propositions 15A and 15B, that every nontrivial facet defining inequality for $P^<$ (for $P^>$) is strongly maximal (strongly minimal).

For $S \subseteq N$, we denote by A^S the submatrix of A consisting of the columns indexed by S.

Theorem 16A. Let $\alpha x \le \alpha_o$ be a maximal valid inequality for $P^<(A,b)$, and let $V := \{j \in N : \alpha_j \neq 0\}$. Then $\alpha x \le \alpha_o$ defines a facet of $P^<(A,b)$ if and only if $\alpha^V x^V \le \alpha_o$ defines a facet of $P^<(A^V,b)$.

Proof. Sufficiency. Suppose $\alpha^V x^V \le \alpha_o$ defines a facet of $P^<(A^V,b)$. Then there exists a set of $|V|$ linearly independent vectors $x^V_{(i)} \in P^<(A^V,b)$ such that $\alpha^V x^V_{(i)} = \alpha_o$, $i = 1,\ldots,|V|$. Extending each $x^V_{(i)}$ to a vector $x^i \in \mathbb{R}^n$ by setting $x^i_j = 0$, $j \in N \setminus V$ (and $x^i_j = x^V_{(i)j}$, $j \in V$) yields $|V|$ linearly independent vectors

$x^i \in \mathbb{R}^n$ such that $\alpha x^i = \alpha_o$, $i = 1, \ldots, |V|$. Further, since $\alpha x \leq \alpha_o$ is maximal, for every $k \in N \setminus V$ there exists some $x \in P^<(A,b)$ such that $x_k = 1$, $x_j = 0$, $j \in N \setminus (V \cup \{k\})$, and $\alpha x = \alpha_o$. It is easy to see that the $n \times n$ matrix X whose rows are these $|V| + |N \setminus V|$ vectors x^i is of the form

$$X = \begin{pmatrix} X_1 & 0 \\ X_2 & I \end{pmatrix}$$

with X_1 nonsingular (of order $|V|$) and I the identity of order $|N \setminus V|$. Thus the rows of X are linearly independent and $\alpha x \leq \alpha_o$ defines a facet of $P^<(A,b)$.

Necessity. Suppose $\alpha^V x^V \leq \alpha_o$ does not define a facet of $P^<(A^V,b)$. Then there exists $\beta^V \in \mathbb{R}_+^{|V|}$, $\beta_o \in \mathbb{R}_+$, such that every $x^V \in P^<(A^V,b)$ satisfies $\beta^V x^V \leq \beta_o$, every $x^V \in P^<(A^V,b)$ such that $\alpha^V x^V = \alpha_o$ satisfies $\beta^V x^V = \beta_o$, and $(\beta^V, \beta_o) \neq \lambda(\alpha^V, \alpha_o)$ for all $\lambda \neq 0$. But then setting $\beta = (\beta^V, 0) \in \mathbb{R}^n$, we obtain an inequality $\beta x \leq \beta_o$ satisfied by all $x \in P^<(A,b)$ and satisfied with equality by all x such that $\alpha x = \alpha_o$, with $(\beta, \beta_o) \neq \lambda(\alpha, \alpha_o)$ for all $\lambda \neq 0$. Thus $\alpha x \leq \alpha_o$ does not define a facet of $P^<(A,b)$. ■

Theorem 16B. Let $\alpha x \geq \alpha_o$ be a minimal valid inequality for $P^>(A,b)$. Then $\alpha x \geq \alpha_o$ defines a facet of $P^>(A,b)$ if and only if $\alpha^V x^V \geq \alpha_o$ defines a facet of $P^>(A^V, b - A^{N \setminus V} e^{N \setminus V})$.

Proof. From Proposition 15B, $\alpha x \geq \alpha_o$ defines a facet of $P^>(A,b)$ if and only if $\alpha y \leq \alpha e - \alpha_o$ defines a facet of $P^<(A,\tilde{b})$; and $\alpha^V x^V \geq \alpha_o$ defines a facet of $P^>(A^V, b' := b - A^{N \setminus V} e^{N \setminus V})$ if and only if $\alpha^V y^V \leq \alpha^V e^V - \alpha_o$ defines a facet of $P^<(A^V, \tilde{b}')$, where $\tilde{b}' := A^V e^V - b' = Ae - b = \tilde{b}$. Thus all we have to show is that $\alpha y \leq \alpha e - \alpha_o$ defines a facet of $P^<(A,\tilde{b})$ if and only if $\alpha^V y^V \leq \alpha^V e^V - \alpha_o$ defines a facet of $P^<(A^V,\tilde{b})$. For the case when $\alpha y \leq \alpha e - \alpha_o$ is maximal, this was established in Proposition 16A. If $\alpha y \leq \alpha e - \alpha_o$ is not maximal, then (since $\alpha x \geq \alpha_o$ is minimal) $\alpha^V y^V \leq \alpha^V e^V - \alpha_o$ is also not maximal, and so neither of them is facet defining. ■

In view of Theorems 16A and 16B, in examining the conditions under which a valid inequality is facet defining for $P^<$ or for $P^>$ we may restrict ourselves to inequalities with strictly positive coefficients. Thus the results of sections 2 and 3 are of general validity for $P^<$ and $P^>$ (i.e. not affected by the restriction to inequalities with nonzero coefficients).

5. Application to Vertex Packing

Let $P^<(G) := P^<(A,e)$ be the *vertex packing polytope* defined on the (undirected) graph $G = (V,E)$ whose edge-vertex incidence matrix is A. In other words, $P^<(G)$ is the convex hull of incidence vectors of vertex packings (independent sets, stable sets) of G. For $S, T \subseteq V$, we denote by (S,T) the set of edges (i,j) such that $i \in S$, $j \in T$, and we write $\alpha(G)$ for the cardinality of a maximum vertex packing (i.e. the independence or stability number) of G. A natural question to ask in connection with $\alpha(G)$ is under what conditions does the (obviously valid) inequality

$$\Sigma(x_j : j \in V) \leq \alpha(G) \tag{13}$$

define a facet of $P^<(G)$? Chvátal (1975) gave the following well known partial answer to this question. Call an edge u of G *α-critical* if $\alpha(G-u) = \alpha(G)+1$, and let E^* be the set of α-critical edges of G. Also, let $G^* := (V,E^*)$.

Theorem 17. (Chvátal 1975) If G^* is connected, then the inequality (13) defines a facet of $P^<(G)$.

This sufficient condition can be weakened by using the concept of an

α-critical cutset, defined as a cutset $(W, V\backslash W)$ such that $\alpha(G(W))+\alpha(G(V\backslash W)) \geq \alpha(G)+1$, where $G(W)$ is the subgraph of G induced by W.

Theorem 18. (Balas and Zemel, 1977). If G has an α-critical cutset $(W, V\backslash W)$ such that for $T := W$ and $T := V\backslash W$

(i) the inequality $\Sigma(x_j: j\varepsilon T) \leq \alpha(G(T))$ defines a facet of $G(T)$; and

(ii) every maximum packing of $G(T)$ is contained in a maximum packing of G;

then the inequality (13) defines a facet of $P^<(G)$.

The sufficient condition of Theorem 18 is not necessary for (13) to be facet defining, as the example below shows.

Using the results of section 2, we will now give a sufficient condition for the inequality (13) to define a facet of $P^<(G)$, which is a direct generalization of Chvátal's above mentioned result, and is weaker than the condition of Theorem 18. The key to this generalization is the following

Remark. An edge $u = (i, j)$ is α-critical if and only if G has two maximum vertex packings S and T, such that $T = (S\backslash\{i\}) \cup \{j\}$.

Proof. If the condition holds, then $S \cup \{j\}$ $(= T \cup \{i\})$ is a vertex packing of $G-u$, hence u is α-critical. Conversely, if u is α-critical, then $G-u$ has a vertex packing Y such that $|Y| = \alpha(G)+1$ and $i, j\varepsilon Y$. Then $S := Y\backslash\{j\}$ and $T := Y\backslash\{i\}$ are both maximum vertex packings in G, with $T = (S\backslash\{i\}) \cup \{j\}$. ■

It is now not hard to see that the presence of an α-critical edge (i, j) implies the existence of a pair $x, y\varepsilon F$ satisfying $x_i = 1-y_i = y_j = 1-x_j$, and $x_k = y_k$, $k\varepsilon V\backslash\{i, j\}$ which is equivalent to condition (3); hence 2-reduction generalizes Chvátal's procedure of constructing the α-critical graph G^* from the case where the subsets N_k of π are singletons, to the more general case where π is a nontrivial partition.

To be specific, let $G^\# := (V, E^\#)$ be the graph constructed as follows:

Initialization. Let S^1 be a maximum vertex packing of G, let $G^1 := (V, E^1)$, $E^1 = \emptyset$, and $t := 1$.

Iterative Step. Given $G^t = (V, E^t)$ with connected components induced by $V_1, \ldots, V_s \subseteq V$, find a maximum vertex packing S^{t+1} of G such that

$$(14) \qquad |S^{t+1} \cap V_k| \begin{cases} \neq |S^t \cap V_k| & k = i, j \\ = |S^t \cap V_k| & k\varepsilon\{1, \ldots, s\}\backslash\{i, j\} \end{cases}$$

for some pair $i, j\varepsilon\{1, \ldots, s\}$. If no such S^{t+1} exists, let $G^\# := G^t$ and stop. Otherwise insert into G^t an edge joining the two components V_i and V_j, i.e. set $E^{t+1} := E^t \cup (u, v)$ for some $u\varepsilon V_i$, $v\varepsilon V_j$, $G^{t+1} := (V, E^{t+1})$, let $t := t+1$ and repeat.

Theorem 19. If $G^\#$ is connected, then $\Sigma(x_j: j\varepsilon V) \leq \alpha(G)$ defines a facet of $P^<(G)$.

Proof. The above procedure amounts to 2-reduction, and $G^\#$ is connected if and only if the procedure stops with the improper partition. Hence the Theorem is a specialization to the vertex packing polytope of Corollary 4. ■

Example 3. Consider the graph G of Figure 1, obtained from the complete 3-partite graph with 5 vertices in each part, $K_{5,5,5}$, by inserting into each of the three parts the edges of a 5-cycle, and deleting the edge sets $(V_1, \{6\})$, $(V_2, \{11\})$, $(V_3, \{1\})$. A maximum vertex packing of G contains two vertices of V_i and one of V_{i+1}, for some $i\varepsilon\{1, 2, 3\}$, or else just one vertex of each V_i, namely 1, 6 and 11. So the inequality

$$\Sigma(x_j: j = 1, \ldots, 15) \leq 3$$

is certainly valid. We want to know whether it defines a facet of $P^<(G)$.

Chvátal's sufficient condition is not satisfied, since the only critical

edges are those of the three 5-cycles and thus G^* is disconnected. Balas and Zemel's sufficient condition is also not satisfied, since G has no critical cutset satisfying (i): although if W stands for the vertex set of a triangle

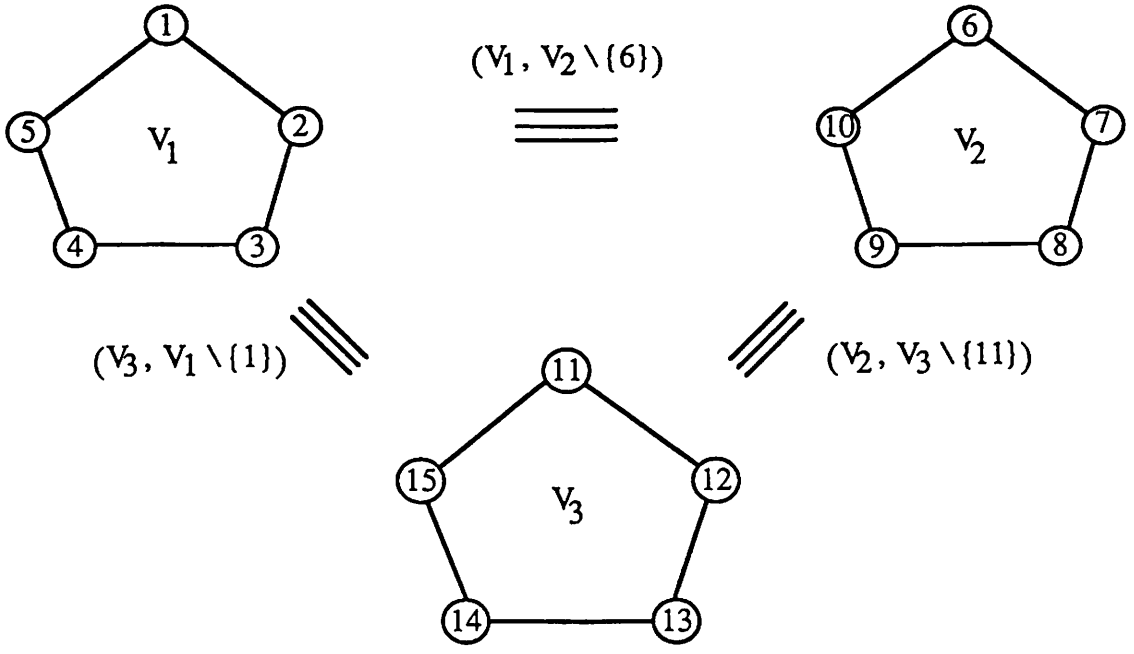


Figure 1

or a pentagon of G the inequality $\Sigma(x_j: j\varepsilon W) \leq \alpha(G(W))$ defines a facet of $P^<(G(W))$, in each of these cases the inequality $\Sigma(x_j: j\varepsilon V\setminus W) \leq \alpha(G(V\setminus W))$ is not facet defining for $P^<(G(V\setminus W))$.

We now apply the procedure that generates $G^\#$. The first four iterations yield the component with vertex set $\{1,\ldots,5\}$ as a path of length 4. The next eight iterations yield two more components (paths of length 4), with vertex sets $\{6,\ldots,10\}$ and $\{11,\ldots,15\}$. Thus G^{13} has three components with vertex sets V_1, V_2 and V_3.

Next consider the maximum vertex packings $S^{13} := \{6,8,11\}$ and $S^{14} := \{1,6,11\}$. We have

$$0 = |S^{13} \cap V_1| \neq |S^{14} \cap V_1| = 1,$$
$$2 = |S^{13} \cap V_2| \neq |S^{14} \cap V_2| = 1,$$
$$|S^{13} \cap V_3| = |S^{14} \cap V_3| = 1,$$

hence condition (14) is satisfied and we add to G^{13} an edge joining a vertex of V_1 to one of V_2 in order to obtain G^{14}. Let the two components of G^{14} have vertex sets $V_1' = \{1,\ldots,10\}$ and $V_2' = \{11,\ldots,15\}$, and consider the maximum vertex packings $S^{14} = \{1,6,11\}$ and $S^{15} = \{11,13,1\}$. Then

$$2 = |S^{14} \cap V_1'| \neq |S^{15} \cap V_1'| = 1$$

and

$$1 = |S^{14} \cap V'_2| \neq |S^{15} \cap V'_2| = 2,$$

hence $G^{15} = G^{\#}$ is obtained from G^{14} by adding an edge joining a vertex of V'_1 to one of V'_2. Since $G^{\#}$ is connected, the inequality $\Sigma(x_j: j\varepsilon V) \leq 3$ defines a facet of $P^<(G)$.■

6. Application to Set Covering

Let A be an mxn 0-1 matrix and b = e. Then $P^>(A,e)$ is the *set covering polytope*, i.e. the convex hull of 0-1 solutions to $Ax \geq e$. A set $S \subseteq N$ is called a *cover* for A if $\Sigma(a_j: j\varepsilon S) \geq e$. The incidence vector of a cover will also be called a cover. If $\beta(A)$ denotes the minimum cardinality of a cover for A, it is of interest to know when the inequality

$$\Sigma(x_j: j\varepsilon N) \geq \beta(A), \tag{15}$$

obviously valid, defines a facet of $P^>(A,e)$. A sufficient condition for this, analogous to Chvátal's condition for the vertex packing polytope, can be stated in the following terms. For $i,j\varepsilon N$, $i \neq j$, denote by A^{ij} the matrix obtained from A by removing all rows $k\varepsilon M$ such that $a_{ki} = a_{kj} = 1$. Call a pair $i,j\varepsilon N$ *β-critical* if $\beta(A^{ij}) = \beta(A)-1$, and let the graph $G^* = (V,E^*)$ have a vertex for every $i\varepsilon N$ and an edge for every β-critical pair $i,j\varepsilon N$.

Theorem 20. (Sassano 1985). If G^* is connected, the inequality (15) defines a facet of $P^>(A,e)$.

Again, the sufficient condition of our Corollary 4 weakens the condition of Theorem 20. To see this, notice the following:

Remark. A pair $i,j\varepsilon N$ is β-critical if and only if there exist minimum covers S and T for A such that $T = (S\backslash\{i\}) \cup \{j\}$.

Proof. If such covers exist, then $S\backslash\{i\}$ $(= T\backslash\{j\}))$ is a cover for A^{ij}, hence the pair i,j is β-critical. Conversely, if the pair i,j is β-critical, then A^{ij} has a minimum cover Y such that $|Y| = \beta(A)-1$ and $Y \cap \{i,j\} = \emptyset$. Then $S := Y \cup \{i\}$ and $T := Y \cup \{j\}$ are minimum covers for A and $T = (S\backslash\{i\}) \cup \{j\}$.■

As in the case of the set packing polytope, we now generate a graph $G^{\#}$ by the following procedure.

Initialization. Let S^1 be a minimum cover for $P^>$, let $G^1 = (V,E^1)$, with $V := N$ and $E^1 := \emptyset$, and $t := 1$.

Iterative Step. Given G^t with connected components induced by $V_1,\ldots,V_p \subseteq V$, find a minimum cover S^{t+1} such that

$$|S^{t+1} \cap V_k| \begin{cases} \neq |S^t \cap V_k| & \text{for } k = i,j \\ = |S^t \cap V_k| & \text{for } k\varepsilon\{1,\ldots,p\}\backslash\{i,j\}, \end{cases} \tag{16}$$

for some pair $i,j\varepsilon\{1,\ldots,s\}$. If no such S^{t+1} exists, let $G^{\#} = G^t$ and stop. Otherwise insert into G^t an edge joining the two components V_i and V_j, i.e. set $E^{t+1} := E^t \cup (u,v)$ for some $u\varepsilon V_1$ and $v\varepsilon V_j$, $G^{t+1} := (V,E^{t+1})$, $t := t+1$, and repeat.

Theorem 21. If $G^{\#}$ is connected, then the inequality (16) defines a facet of $P^>(A,e)$.

Clearly, Theorem 21 is a specialization of Corollary 4. On the other hand, the condition of Theorem 20 asserts that (16) is facet defining if the graph $G^{\#}$ obtained by restricting the Iterative Step of our procedure to pairs $i,j\varepsilon\{1,\ldots,p\}$ such that V_i and V_j are *singletons*, is connected. Hence the

sufficient condition of Theorem 21 is a weakening of the one in Theorem 20.

Acknowledgements

Thanks are due to Matteo Fischetti for useful comments on an earlier draft of this paper.

References

A. Bachem and M. Grötschel (1982), "New Aspects of Polyhedral Theory." B. Korte (ed.), *Modern Applied Mathematics, Optimization and Operations Research.* North Holland, 51-106.

E. Balas and E. Zemel (1977), "Critical Cutsets of Graphs and Canonical Facets of Set Packing Polytopes." *Mathematics of Operations Research, 2,* 15-20.

V. Chvátal (1975), "On Certain Polytopes Associated with Graphs." *Journal of Combinatorial Theory, B, 18,* 138-154.

J. Edmonds, L. Lovász and W. R. Pulleyblank (1982), "Brick Decompositions and the Matching Rank of Graphs." *Combinatorica, 2,* 247-274.

G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization.* Wiley, 1988.

W. R Pulleyblank, "Polyhedral Combinatorics," in A. Bachem, M. Grötschel and B. Korte (Eds.), *Mathematical Programming: The State of the Art,* Springer, 1983.

A. Sassano, "On the Facial Structure of the Set Covering Polytope." IASI-CNR Report No. 132, Rome, 1985.

A. Schrijver, *Theory of Linear and Integer Programming.* North Holland, 1986.

Some Provably Hard Crossing Number Problems

Daniel Bienstock
Columbia University
New York, N.Y. 10027

Abstract

This paper presents a connection between the problem of drawing a graph with minimum number of edge crossings, and the theory of arrangements of pseudolines, a topic well-studied by combinatorialists. In particular, we show that any given arrangement can be forced to occur in every minimum crossing drawing of an appropriate graph. Using some recent results of Goodman, Pollack and Sturmfels, this yields that there exists no polynomial-time algorithm for producing a straight-line drawing of a graph, which achieves minimum number of crossings from among all such drawings. While this result has no bearing on the P vs. NP question, it is fairly negative with regard to applications. We also study the problem of drawing a graph with polygonal edges, to achieve the (unrestricted) minimum number of crossings. Here we obtain a tight bound on the smallest number of breakpoints which are required in the polygonal lines.

1. Introduction. A *drawing* of a simple graph G is a subset of the plane $\mathbb{R}^2$ where each vertex is represented by a different point, and each edge is represented by a homeomorph of the closed unit interval I^1 with appropriate ends. Further, the drawings of any two edges meet at most once, and if they do, then either the two edges are incident to a common vertex, where their drawings meet, or the two drawings cross at their intersection point (the term cross is assumed to be understood). The *crossing number problem* consists of producing a drawing of G which achieves the least possible number of crossings (this parameter is called the *crossing number* of G and denoted cr(G)). This problem is of interest in VLSI theory and in wiring layout problems (see [Le]), and it has long been of interest in the graph theory community (see [EG1], [Th2], [Tu]). Computing the crossing number is NP-hard [GJ] (and the decision problem is NP-complete).

In many of the applications, further, it is desirable that the edges be drawn as straight-line segments, with no restriction to orientation. Such a drawing has classically been called *rectilinear*, and the minimum number of crossings in a rectilinear drawing is called the *rectilinear crossing number*, denoted $cr_1(G)$. We remark that practically every paper on crossing numbers has in fact also dealt with rectilinear crossing numbers (the latter being used to study the former). Using the proof in [GJ], it can be shown that computing the rectilinear crossing number is NP-hard. While this problem is not yet known to be in NP, it is clear that the (cartesian) coordinates of the vertices in a drawing can be assumed to be rational, and thus, integral. A generalization of the rectilinear crossing number is the *t-polygonal* crossing number (where $t \geq 1$ is an integer) denoted $cr_t(G)$, which arises when every edge must be drawn as a t-polygonal line, i.e., a polygonal line with at most t segments. Thus t=1 yields the rectilinear case. We will see below the motivation for studying t-polygonal drawings of graphs with $t > 1$.

In this paper we study the connection between crossing number problems and the theory of arrangements of pseudolines, an area of high interest in combinatorial geometry, which has most notably been studied by Grünbaum, Goodman, Pollack and others (see [Gr], [Go], [GP1-6], [GPS]). For the purposes of this paper, a *pseudoline* will be a homeomorph in $\mathbb{R}^2$ of the closed unit interval I^1. An *arrangement* A of pseudolines is a collection of pseudolines, every two meeting at exactly one interior point, where they cross, and such that the ends of each pseudoline are in the closure of the unbounded component of $\mathbb{R}^2 \backslash A$ (with the obvious abuse of notation here). Further, all arrangements considered here will be *simple*, meaning that no three of the pseudolines meet at a common point. Following Grünbaum [Gr] we notice that an arrangement defines a 2-dimensional cell complex (the unbounded face can be disregarded here). Two arrangements are said to be *isomorphic* if the labeled cell complexes they define are isomorphic (in which case we say one *realizes* the other, or is a *realization* of the other). A *t-polygonal realization* of an arrangement is one where each pseudoline is t-polygonal. Thus, a 1-polygonal realization is a straight-line realization.

Our first result is as follows (stated in abridged form here):

Theorem 1. Let A be an arrangement of n pseudolines. There is a graph G_A, so that

(i) Every drawing of G_A with $cr(G_A)$ crossings contains a realization of A. Further, $cr(G_A)=5n(n-1)$ and $|E(G_A)| = O(n^3)$.

(ii)Let $t \geq 1$. If A has a t-polygonal realization, then every t-polygonal drawing of G_A with $cr_t(G_A)$ crossings contains a t-polygonal realization of A. Further, $cr_t(G_A)=5n(n-1)$ and $|E(G_A)| = O(tn^4)$ ($O(n^3)$ for t=1).∎

Thus if we can construct arrangements all of whose realizations are "bad" in some technical sense, we will also have graphs, all of whose minimum-crossing drawings are also "bad" in the same sense. In particular, recent work of Goodman, Pollack and Sturmfels implies the following remarkable result (via dualization to straight-line arrangements):

Theorem [GPS]. For any $n \geq 1$ there exists an arrangement E^n of straight-line segments, such that in every straight-line realization of E^n the coordinates of the ends require exponentially many bits. ∎

Together with Theorem 1 (ii) (with t=1), this yields:

Theorem 2. There exists an infinite family of graphs $\{G^n\}$, such that in every rectilinear drawing of G^n with $cr_1(G^n)$ crossings, the coordinates of the vertices require more than polynomially many bits. ∎

As a result, there does not exist a polynomial-time algorithm for producing a rectilinear drawing of a graph which achieves the rectilinear crossing number. Here we are assuming a model where either (a) The coordinates of the vertices must be written down, or (b) A physical drawing of the graph must be produced, and drawing a graph consumes resources (such as time or space) proportional to the size of the drawing (e.g. if the graph is to be drawn on a computer screen). In a different model of graph drawing our result might not have any impact at all. Clearly, the result does not imply $P \neq NP$, for the simple reason that there may be a polynomial-time algorithm for *computing* the rectilinear crossing number, which does not rely on *drawing* the graph. In fact such an algorithm could not even write the coordinates of all vertices.

Let $t \geq 1$. As shown in [BD] using t=2 instead of t=1 can dramatically decrease the number of crossings (the improvement in fact cannot be bounded as a function of the size of the graph). However, for any fixed t there is a result similar to Theorem 2 (given in Theorem 3). On the other hand, already with t=2, in polynomial space (logarithmically many bits) one can get within a quadratic bound of the crossing number.

How complicated can a minimum-crossing drawing of a graph be? In other words, is there a fixed number t, so that for all graphs G, the crossing number of G can be achieved with a t-polygonal drawing? We show that the answer is no, and thus, for graphs of crossing number k or less, the minimum possible such t depends on k; let us call it t(k). We prove:

Theorem 4. There exist constants c_1 and c_2, so that for every $k \geq 1$,

$$c_1 k^{1/2} \leq t(k) \leq c_2 k^{1/2}. \blacksquare$$

To obtain the lower bound, we apply a recent result of Kratochvil and Matousek [KM], and our construction in Theorem 1. The proof of the upper bound relies on a direct construction.

2. Forcing arrangements in drawings of graphs.

In this section we prove Theorem 1. For the proof we will need a definition. Let G be a planar graph, and $k > 1$ an integer. The *k-thickening* G' of G is the planar graph obtained by replacing each edge e={u,v} of G by a set b(e) of k edge-disjoint paths of length two, with ends u and v. If $e \neq e'$ it is assumed that the set of internal vertices of the paths in b(e) is disjoint from the similar set in b(e'). So if the paths in each b(e) are drawn very close together then G' will indeed be planar.

Theorem 1. Let A be an arrangement of n pseudolines. There exists a graph G_A with $cr(G_A)=5n(n-1)$ and $|E(G_A)| = O(n^3)$, with a distinguished subset S_A of edges, such that

(i)In every drawing of G_A with $cr(G_A)$ crossings the drawing of S_A contains a realization of A. Further, $cr(G_A)=5n(n-1)$ and $|E(G_A)| = O(n^3)$.

(ii)Let $t \geq 1$. If A has a t-polygonal realization, then in every t-polygonal drawing of G_A with $cr_t(G_A)$ crossings the drawing of S_A contains a t-polygonal realization of A. Here $cr_t(G_A)=5n(n-1)$ and $|E(G_A)| = O(tn^4)$ ($O(n^3)$ for t=1).

Proof: (i)The graph G_A is obtained in several steps. See Fig. 1 for an example.
1. We begin by replacing each $x \in A$ with two copies, x_1 and x_2 drawn very close to each other. Next, we obtain a planar graph L by placing a vertex at crossing and at each end of every pseudoline. Thus, for each $x \in A$, L contains two edge disjoint paths $p_1(x)$ and $p_2(x)$, where $p_i(x)$ has ends (degree one vertices) $u_{i1}(x)$, $u_{i2}(x)$. We assume that the labeling has been done so that, as we traverse the outer facial boundary of L counterclockwise, $u_{21}(x)$ immediately follows $u_{11}(x)$, $u_{22}(x)$ appears after $u_{21}(x)$, and $u_{12}(x)$ immediately follows $u_{22}(x)$, see Fig. 1 (b)). Next, we add to L a cycle C, joining all the vertices $u_{ik}(x)$ in the cyclic order, so as to form with L a planar graph. Further, C contains a vertex $v_1(x)$ between every two vertices $u_{11}(x)$, $u_{21}(x)$, and a vertex $v_2(x)$ between every two vertices $u_{22}(x)$, $u_{12}(x)$ $x \in A$.
2. For each $x \in A$, we add an edge e(x) with endpoints $v_1(x)$ and $v_2(x)$, and so that e(x) is drawn inside C and "between" $p_1(x)$ and $p_2(x)$. Let S be the set of all edges e(x), $x \in A$, and $H = L \cup C \cup S$. See Fig. 1 (c).
3. Take a copy $H' = C' \cup L' \cup S'$ of H, drawn outside H (and with the obvious notation), and a matching M joining the vertices of C to those in C', so that $L \cup C \cup L' \cup C' \cup M$ is planar. We replace $C \cup C' \cup M$, by its m-thickening W (where $m = 5n(n-1)+1$) so that the edges of W have no crossings at all. The resulting graph is G_A, and we set $S_A = S \cup S'$. See Fig. 1(d)

Let D_A be the drawing we have just constructed, which has $5n(n-1)$ crossings. Suppose now that D* is a drawing of G_A with $cr(G_A) \leq 5n(n-1)$ crossings. Consider an edge t of $C \cup C' \cup M$. Since $m > cr(G_A)$, it follows that at least one of the paths in b(t) has no crossings in D*. Thus we obtain a planar drawing of $C \cup C' \cup M$ in the obvious way. Since $C \cup C' \cup M$ is 3-connected, it has a unique embedding, and its drawings in D* and D_A are homeomorphic in the sphere. So in D* both C and C' bound faces of $C \cup C' \cup M$, which must contain the drawings of $C \cup L$ and $C' \cup L'$ (respectively). Next, let $x,y \in A$. Then, by construction, the endpoints of e(x) and e(y) alternate along C. In fact, the endpoints of e(x) and the ends of each path $p_i(y)$, i=1,2, alternate along C (and

similarly, the endpoints of e(y) and the ends of each path $p_i(x)$, i=1,2, alternate along C). Since for any $z \in A$, $p_1(z)$ and $p_2(z)$ are edge-disjoint, we count 5 crossings in D* corresponding to the pair x,y (the Jordan curve theorem). In this way we count 5n(n-1)/2 crossings in the region bounded C. Similarly with C'. Thus $5n(n-1) = cr(G_A)$, and the 5n(n-1)/2 crossings we have just counted are all the crossings in the region bounded by C (resp. C'). Either C or C' bound an inner face of $C \cup C' \cup M$ in D*, say C does. Consequently, in D*, (i) the drawing of $C \cup L$ is planar, and (ii) For each $x \in A$, e(x) does not cross any edge in $p_i(x)$, i=1,2. Then (ii) implies that for each $x \in A$, e(x) is drawn "between" $p_1(x)$ and $p_2(x)$. We conclude that the drawing of S realizes A, as desired. This concludes the proof of (i).

(ii) Assume that the pseudolines in A are t-polygonal. The proof of (ii) proceeds exactly as that of (i), with the exception that the initial drawing D_A must now, in addition, be t-polygonal, so that we can argue that $cr_t(G_A) \leq 5n(n-1)$. It is clear that the edges of L and S can be assumed to be t-polygonal. The same holds for the edges of C, by drawing them "following" the outer facial cycle of L. Further, the edges of the matching M can be drawn as O(tn)-polygonal lines. So let us replace each edge of M by a path of length O(tn), obtaining a graph N. The planar graph $C \cup L \cup C' \cup L' \cup N$, although not 3-connected, still has a unique embedding on the sphere (the only exception to 3-connectedness are the degree-2 vertices in N). From now on, we can proceed with the m-thickening of $C \cup C' \cup N$, and the rest of the proof is as above. Notice that $|E(G_A)| = O(nm)O(tn) = O(tn^4)$. To obtain the reduction to $O(n^3)$ for the case t=1, notice that in this case, the region enclosed by the cycle C can be assumed to be a rectangle. Thus in this case the edges of M need only be subdivided (at most) four times each, which yields the desired bound. This concludes the proof of (ii). ∎

Comment: The graph G_A has vertices of large degree, but the same result can be achieved with a graph of maximum degree 3, although the proof is somewhat longer. The construction relies on using large section of a "wall", a special type of cubic graph.

3. Graphs which require more than polynomially many bits.

Let us apply Theorem 1 (ii), with t=1, where for A we use one of the straight-line arrangements obtained from the results in [GPS]. We will have:

Theorem 2. There exists an infinite family of graphs $\{G^n\}$, such that in every rectilinear drawing of G^n with $cr_1(G^n)$ crossings, the coordinates of the vertices require more than $\exp(c|E(G^n)|^{1/3})$ many bits, where c is a fixed constant. ∎

Can the situation in Theorem 2 be avoided, if we use t-polygonal drawings, where $t > 1$ is "small"?. The answer is no, as shown in Theorem 3 given below. The proof of this theorem makes use of a certain arrangement of nine pseudolines due to Ringel [Ri] which is described in [Gr] (also see [GP2]). This arrangement, which we denote by R_9, does not have a straight-line realization, but on the other hand it has 2-polygonal realizations in which only one pseudoline is not 1-polygonal (i.e., all the others are straight-lines. The distinguished pseudoline can be arbitrarily chosen).

Theorem 3. Let $t>1$. There exists an infinite family of graphs $\{G^{n,t}, n=1,2,....\}$, such that every t-polygonal drawing of $G^{n,t}$ with $cr_t(G^{n,t})$ crossings requires more than polynomially many bits.

Proof: The proof of this (perhaps not unexpected) theorem requires an intermediate construction. Let M denote an arbitrary arrangement of n straight-lines. For $t = 1,2,...$, we will inductively define an arrangement A(t) of t-polygonal pseudolines as follows. First, let A(1) be a straight-line realization of M. Suppose now that $t > 1$. Replace each pseudoline $x \in A(t-1)$ with a set S(x) of 9 copies, drawn very close to one another, and so that the copies each segment of x are parallel. The "spacing" between consecutive copies of x is determined below.

We obtain a 2-polygonal realization of R_9 as follows. First, choose an arbitrary $y \in S(x)$, and stretch it a small distance beyond the ends of the other members of S(x) (we still call the stretched pseudoline y for convenience). Next, attach to the end of each pseudoline z in S(x) an additional (small) straight-line segment e(z). Let r(y) be the 2-polygonal line formed by e(y) and the stretched portion of y. It is not hard to see that if (1) the spacing between copies of x, (2) the stretching of y, and (3) the choice of segments e(z), are properly carried out, then the arrangement $\{e(z) : z \neq y\} \cup \{r(y)\}$, will be a 2-polygonal realization of R_9. See Fig. 2. Here r(y) is the only pseudoline in this 2-polygonal realization of R_9, which is not a straight-line. Notice that for each pseudoline in S(x) we introduced at most one new

breakpoint. Thus the pseudolines $z \cup e(z)$ are all t-polygonal, since x is (t-1)-polygonal

The arrangement which results from carrying out this operation, for every x, is A(t), which is t-polygonal, as desired. The following claim is clear and we include the proof for completeness.

Claim. Every t-polygonal realization of A(t) includes, as a subdrawing, a realization of E^n.

Proof of Claim. The claim is clear for t=1. Next, assume that $t > 1$, and consider any t-polygonal realization L of A(t). Denote by $R_9(x)$ the portion of the pseudolines in S(x) which realize R_9. Clearly, for every $x \in A(t-1)$, there is a pseudoline $z(x) \in S(x)$, which has in L at least one breakpoint in the portion contained in $R_9(x)$ (otherwise $R_9(x)$ would have a straight-line realization). Thus the arrangement $\{z(x): x \in A(t-1)\}$ includes in L a (t-1)-polygonal realization of A(t-1), and the claim follows by induction.
This completes the proof of the claim.

We note now that $|A(t)| = 9^t n$. To conclude the proof of Theorem 3, let us use for M the "bad" arrangement E^n used in Theorem 2. Then we will have that in every t-polygonal realization of A(t) the coordinates of the ends and breakpoints require more than polynomially many bits. Hence, we apply the construction in Theorem 1 (ii) to A(t), and define $G^{n,t} = G_{A(t)}$. ∎

The following question arises as a result of Theorems 2 and 3: what can be achieved in polynomial space? In other words, given a graph G and fixed t, can we always provide a t-polygonal drawing of G, in polynomial space, with a number of crossings bounded as a function of $cr_t(G)$? For t=2, the answer is yes, in a strong sense, as follows. In [BD], the authors showed:

Theorem [BD]. For every graph G, $cr_2(G) \leq 2(cr(G))^2$. ∎

In fact, if G has n vertices, the construction in [BD] yields a 2-polygonal drawing of G, in a grid of size $O(n(cr(G))^{1/2})$, with at most $2(cr(G))^2$ crossings (we stress that this construction carries out something more difficult than what we asked for: it approximates cr(G), rather than $cr_2(G)$). This result is based in the theorem of deFraisseix, Pach and Pollack [FPP], that any planar graph has a straight-line drawing of linear size (see also Schnyder [Sc]). Essentially, the desired 2-polygonal drawing of G is obtained from a planar drawing of a graph with O(n) vertices. We refer the reader to [BD] for details.

Thus, for the case t=2, the above question has a positive answer, in a strong sense. Nevertheless, it is an open question whether the quadratic bound is best possible for t=2, and whether it can improved for larger t. Moreover, the case t=1, perhaps the most interesting, is completely open.

As one last application of Theorem 1, we point out that the method used in the proof of Theorem 3 easily yields, for every $t > 1$, a graph H with $cr(H) = cr_t(H) < cr_{t-1}(H)$. By taking disjoint copies of such graphs, one obtains:

Corollary. For every $t > 1$ there exists a graph G, with $cr(G) = cr_t(G)$, and such that for every $t \geq i > 1$, $cr_i(G) < cr_{i-1}(G)$. ∎

4. Achieving the crossing number with t-polygonal drawings.

In this section we study the following question: is there a fixed number t, so that for all graphs G, $cr_t(G)=cr(G)$? Suppose first that we are only concerned with graphs G with bounded crossing number, say, at most r. Then, from a drawing of G with cr(G) crossings, we obtain a drawing of a planar graph by placing a vertex at each crossing. Passing to a straight-line planar drawing of this graph, and removing the added vertices, we will obtain an r-polygonal drawing of G which attains the crossing number. But what if cr(G) is no longer bounded? Then, as will be shown below, the smallest value t so that $cr_t(G)=cr(G)$ will in general depend on cr(G). More formally, let

$$\chi(k) = \{ G : cr(G) \leq k \}, \quad \text{and}$$

$$t(k) = \min \{ t : cr_t(G) = cr(G) \text{ for all } G \in \chi(k) \}.$$

Then we have:

Theorem 4. There exist constants c_1 and c_2, so that for every $k \geq 1$,

$$c_1 k^{1/2} \leq t(k) \leq c_2 k^{1/2}. \blacksquare$$

The rest of the paper is devoted to the proof of Theorem 4.

4.1. The lower bound.

In order to obtain the lower bound, we will use the following theorem, recently proved by Kratochvil and Matousek.

Theorem [KM]. For each n, there exists an arrangement Z(n) of n pseudolines, which cannot be realized with t-polygonal pseudolines unless $t \geq cn$, where c is a constant. ∎

[Remark: in [KM], arrangements are allowed to have pairs of pseudolines that do not meet, but this detail is easily dealt with to obtain the above theorem].

Now given n, consider the graph $G_{Z(n)}$ produced by Theorem 1. Write $H = G_{Z(n)}$. We have that $cr(H)=O(n^2)$. On the other hand, unless $t \geq cn$, we also have $cr_t(H) > cr(H)$ by definition of Z(n). Thus the lower bound in Theorem 4 is proved.

4.2. The upper bound.

In this section we will prove the upper bound in Theorem 4, $t(k) \leq O(k^{1/2})$. Let D be a drawing of G with cr(G) = k crossings. We partition the edges of G into (at most) two classes, H and L, where

H = set of edges with more than $2k^{1/2}$ crossings in D, and

L = set of edges with at most $2k^{1/2}$ crossings in D.

(For convenience, we will also use H and L to refer to the corresponding subgraphs of G). Now if H is empty, then we are essentially done: we obtain, from D, a planar graph by placing a new vertex at each crossing point, in addition to the vertices of G. Since any planar graph has a straight-line drawing (see [Th2] for a short proof), we will obtain a $(2k^{1/2})$-polygonal drawing of G with cr(G) crossings, as desired. In general, H is of course nonempty, but still this basic construct is the appropriate idea to use.

Our procedure is to first draw H, and then L, always obtaining drawings homeomorphic to those in D. The key fact here is that $| H | < k^{1/2}$. So if we draw H, ignoring L, and using the planar graph construction as in the previous paragraph, the members of H will be $(k^{1/2} + 1)$-polygonal lines (we stress that the crossings of edges in H with edges in L are ignored here). Similarly, if we draw L, even taking into account the crossings with edges in H, and again using the planar graph method, the edges will be $O(k^{1/2})$-polygonal lines (by definition of L). As shown below, by building more structure especially into the drawing of H, we will be able to piece together the two drawings as desired.

Step 1. Drawing H.

Consider the drawing of H provided by D. In general, this drawing will have several arc-connected components, which we call *pieces*. Each piece consists of possibly more than one graph-theoretic components of H, and each edge in H appears in one piece. Each edge is partitioned into *sections*, where a section is the portion of the edge between consecutive crossings (with other edges of H) or a portion between a vertex and the crossing closest to it. We will draw each piece separately, with some added structure.

Thus, let Z be a piece. Then the complement in $\mathbb{R}^2$ of Z consists of several connected regions, or *faces* (so if Z is planar these are faces in the standard sense). Each face is bounded by sections, although the boundary may not be simple: both sides of a section may be incident to the same face. One of the faces is unbounded, in the sense that it is homeomorphic to the complement of a closed disk. From Z, obtain a planar graph as follows. First, put one vertex at each crossing point (once more, these are crossings involving two edges in H), in addition to the original vertices. Thus each section becomes an edge. Next, subdivide each section s by introducing one new vertex v(s). Finally, for each face F, draw a vertex w(F) in F. Then we join w(F) to each vertex v(s) (where s is a bounding segment of F) with a length-two path. If the boundary of F is not simple, and if both sides of a segment s are incident to F, then we use *two* length-two paths from v(s) to w(F), drawn to each side of s (see Fig. 3). All of the length-two paths are edge-disjoint, and can be drawn without crossings. We obtain a planar graph Z'.

We note that the outer face of Z' contains some vertex w(F). By adding edges if necessary, we may assume this outer face is a triangle. We produce a drawing of Z by taking a straight-line drawing R(Z) of Z'. It is clear that this drawing of Z is $O(k^{1/2})$-polygonal.

Step 2. Beginning the drawing of L.

In order to draw L, let us return to the original drawing D of G. The complement of H is partitioned into several connected regions, which we call *plots*. Each plot is the intersection of several faces of different pieces, as discussed above. In general, each plot P will be incident to several pieces, all of whom, with at most one exception, it encloses (and the one exception encloses P). In other words, P is homeomorphic to an open disk (or to $\mathbb{R}^2$) with several holes cut out (thus each boundary component of P corresponds to a different piece). We will draw L by separately drawing the part of L contained

in each plot. Below we will see how to piece together all of these subdrawings and also the drawing of H produced above.

Thus, let P be a plot, and assume that P contains part of the drawing of L in D. Let us "expand" each boundary component slightly, to obtain a larger disk. Thus (for example) if Z is a piece enclosed by P and incident to it, then we will obtain a disk that encloses Z "very closely" (see Fig. 4 (a,b)). We note that this expansion will "truncate" portions of edges of L that appear in P and intersect Z (we say such portions *link* Z and P). Let P' be the subset of P resulting from the removal of the expanded boundary components. We assume that the expansions are small enough, so that all vertices and crossing points contained in P are also contained in P'.

Consider the subdrawing of L contained in P'. From this subdrawing, obtain a planar graph by placing a vertex at each crossing point, in addition to any vertices of G. Further, if Z is a piece, the subset in P' of each portion that links Z and P, terminates at some point along the expanded boundary of Z. At each such termination point, we place one new vertex, called a termination vertex (so at this stage, each termination vertex has degree one). Finally, we add an additional vertex v(Z) and edges joining v(Z) to all termination vertices corresponding to Z (so now each termination vertex has degree two). As noted above, there is at most one piece Z which encloses P. If such is the case, the planar graph we have constructed so far has the vertex v(Z) in its outer face. We triangulate this face, making sure that v(Z) remains in the outer face. This concludes the construction of the planar graph corresponding to P (See Fig. 4 (c)). Let S(P) denote a straight-line drawing of this planar graph (for convenience, we will use S(P) to refer to the graph itself).

We remark that, for every piece Z incident to P, the linking portions of Z and P intersect the boundary of Z with a certain clockwise. This is mirrored by the clockwise ordering of the corresponding edges incident to v(Z).

Step 3. Putting L and H together.

The notion of enclosure can be used to define a partial order on the plots (A > B if A encloses B. It is not difficult, but tedious, to actually prove this is a partial order). We will put together the various drawings S(P) and R(Z) by moving downwards in the partial order.

Thus, let P be a plot maximal in the partial order. Start with the graph S(P). Consider first a piece Z enclosed by P and incident to it. Let us draw a very small triangle around v(z), and remove its interior (so the edges incident

to v(z) in S(P) are truncated). Inside the resulting triangle we place a small copy of the drawing R(Z). We recall that R(Z) contains a certain vertex w(F), contained in the outer face of F, which is connected (via length-two paths) to the vertices v(s), for all boundary sections s of Z. Further, the outer face of R(Z) is a triangle.

Next we draw the portions that link P and Z. Consider any given linking portion i, that (say) terminates on some boundary segment s of Z. Now corresponding to i, there was some edge e_i incident to v(Z), which is now truncated by the small triangle around v(Z). The idea is that we now extend the edge e_i, *into* the triangle, to a point arbitrarily close to w(F), and then along the appropriate length-two path in R(Z) to the section s (See Fig. 5 (a-d)).

Can we simultaneously carry this out for all linking sections i of P and Z, without introducing new crossings? The answer is yes, and there are two facts that ensure this: (1) The clockwise ordering of the linking sections along the boundary of Z is isomorphic to the clockwise ordering of the edges incident to v(z), in S(P), and (2) P' contains all crossing points of edges of L that were contained in P. Thus, in fact, the simultaneous drawing can be carried out, which yields a drawing of all the linking sections that is homeomorphic to that in D.

It is clear that the drawings of the linking portions can be done with polygonal lines. But how many breakpoints are involved? First of all, to go from the boundary of the triangle enclosing v(Z), to the vicinity of w(F), we only need O(1) breakpoints (we wrap around the triangle until we get to a point that sees w(F), and then we extend from that point to w(F)). Further, we can go from w(F) to each boundary section s with at most O(1) additional breakpoints. Thus we overall need O(1) breakpoints per linking portion. Notice that each boundary section s will in general contain points of several linking portions, in some permutation.

This concludes the description of how to put together the drawings R(Z) and S(P), in the case where P encloses Z. The reverse case is simpler and similar, and is left to the reader (here we would take a small copy of S(P), and place it "at" the corresponding vertex w(F)).

Having handled P, then we recursively handle the other plots, by moving down the partial order given by enclosure. The "freedom" mentioned at the end of the next-to-last paragraph guarantees that the recursive process

can indeed be carried out, yielding at the end a polygonal drawing of G with cr(G) crossings.

Counting breakpoints.

We saw in Step 1 that every edge of H is drawn as an $O(k^{1/2})$-polygonal line. How about the edges of L? These edges have two types of breakpoints: (1) Those produced when drawing the graphs S(P), and (2) Those produced when drawing the linking sections. Consider an edge e of L. Then the total number of type (1) breakpoints on this edge is at most of the order of the number of crossings on this edge (recall how S(P) was constructed), which is $O(k^{1/2})$ by definition of L. Each linking section contained in e also contributes O(1) breakpoints. But for each linking section we also count an additional crossing on e (or an endpoint of e). Thus again we count at most $O(k^{1/2})$ additional breakpoints. Hence every edge is drawn as an $O(k^{1/2})$-polygonal line, as desired.

This concludes the proof of the upper bound $t(k) \leq O(k^{1/2})$.

5. Future problems.

There are several open problems that are worth mentioning here. One has already been mentioned above; namely, the approximation of t-polygonal crossing numbers in polynomial space, or the approximation of the crossing number using t-polygonal drawings with t small *and* in polynomial space. Another intriguing possibility is the use of systems of polynomials (similar to those used in the lower bound in Theorem 4) to drive polynomial-time heuristics for approximation of crossing numbers. Such heuristics would rely on approximate algorithms for solving systems of polynomial inequalities; a problem that has itself been recognized as very difficult by various researchers. Yet another possible tool to approximate the crossing number would be to use, instead of t-polygonal lines, polynomial lines (graphs of polynomials in one variable) of small degree. See [GP5] for results on realizations of arrangements of pseudolines with polynomial lines. From the point of view of wiring applications, a perhaps more useful approach would be to use splines of polynomials.

Acknowledgment. I would like to thank N. Alon, J.E. Goodman and R. Pollack for useful discussions.

References

[BD] D. Bienstock and N. Dean, Bounds on rectilinear crossing numbers, submitted.

[EG1] P. Erdös and R.K. Guy, Crossing number problems, Amer. Math. Monthly 80 (1973), 52-58.

[GJ] M.R. Garey and D.S. Johnson, Crossing number is NP-complete, SIAM J. Alg. Disc. Meth. 4 (1983), 312-316.

[FPP] H. deFraysseix, R. Pollack, and J. Pach, Small sets supporting Fáry embeddings of planar graphs, Proc. 20^{tht} STOC (1988), 426-433.

[Go] J.E. Goodman, Proof of a conjecture of Burr, Grünbaum and Sloane, Disc. Math. 32 (1980), 27-35.

[GP1] J.E. Goodman and R. Pollack, On the combinatorial classification of nondegenerate configurations in the plane, J. Comb. Theory, Ser. A 29 (1980), 220-235.

[GP2] J.E. Goodman and R. Pollack, Proof of Grünbaum's conjecture on the stretchability of certain arrangements of pseudolines, J. Comb. Theory, Ser. A 29 (1980), 385-390.

[GP3] J.E. Goodman and R. Pollack, Multidimensional sorting, SIAM J. Comp. 12 (1983), 484-501.

[GP4] J.E. Goodman and R. Pollack, Semispaces of configurations, cell complexes of arrangements, J. Comb. Theory, Ser. A 37 (1984), 257-293.

[GP5] J.E. Goodman and R. Pollack, Polynomial realization of pseudoline arrangements, Comm. Pure and Appl. Math. XXXVIII (1985), 725-732.

[GP6] J.E. Goodman and R. Pollack, There are asymptotically fewer polytopes than we thought, Bull. AMS 14 (1986), 127-129.

[GP7] J.E. Goodman and R. Pollack, Upper bounds for configurations and polytopes in $\mathbb{R}^d$, Disc. Comp. Geom. 1 (1986), 219-227.

[GPS] J.E. Goodman, R. Pollack and B. Sturmfels, Coordinate representation of order types requires exponential storage, Proc. 21st STOC (1989), 405-410.

[Gr] B. Grünbaum, Arrangements and spreads, CBMS Regional Conference Series in Applied Mathematics 10, Amer. Math. Soc., Providence, RI (1972).

[KM] J. Kratochvil and J. Matousek, Intersection graphs of segments, submitted (manuscript 1989).

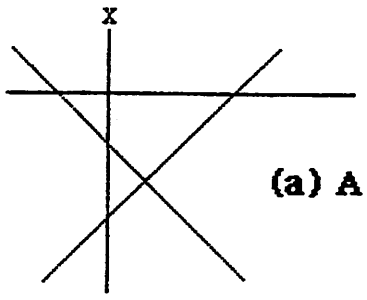
[Le] F.T. Leighton, "Complexity issues in VLSI," The MIT Press, Cambridge (1983).

[Ri] G. Ringel, Teilungen der Ebene durch Geraden oder topologische Geraden, Math. Z. 64 (1956), 79-102.

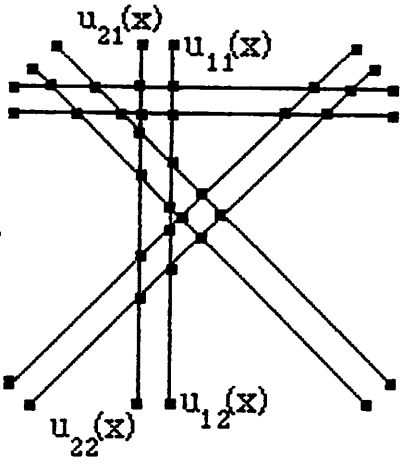
[Sc] W. Schnyder, Embedding planar graphs on the grid, Preliminary report, Louisiana State University, Baton Rouge.

[Th2] C. Thomassen, Rectilinear drawings of graphs, J. Graph Theory 12 (1988), 335-341.

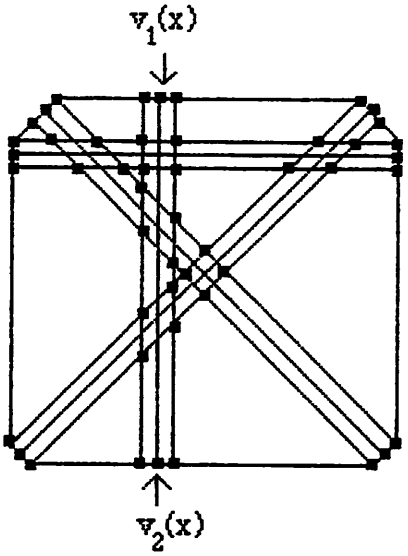
[Tu] W.T. Tutte, Toward a theory of crossing numbers, J. Combin. Theory 8 (1970), 45-53.



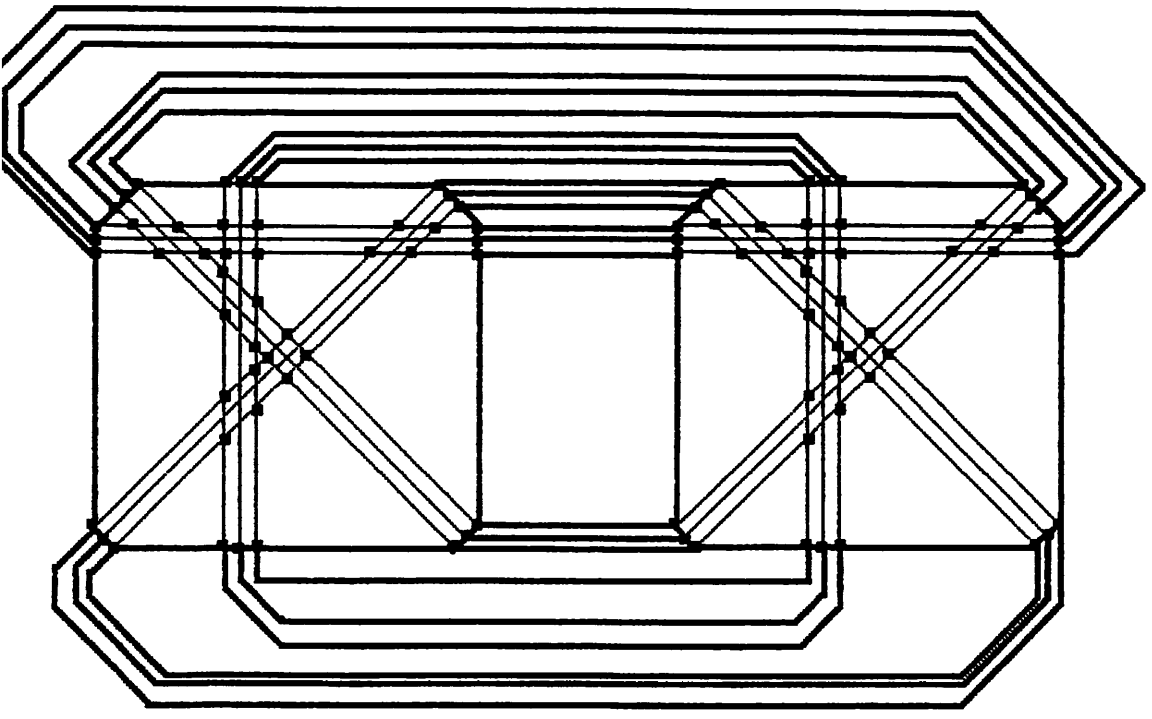
(a) A

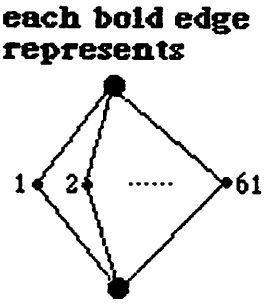


(b) L



(c) L U C U S





(d) G_A

FIGURE 1

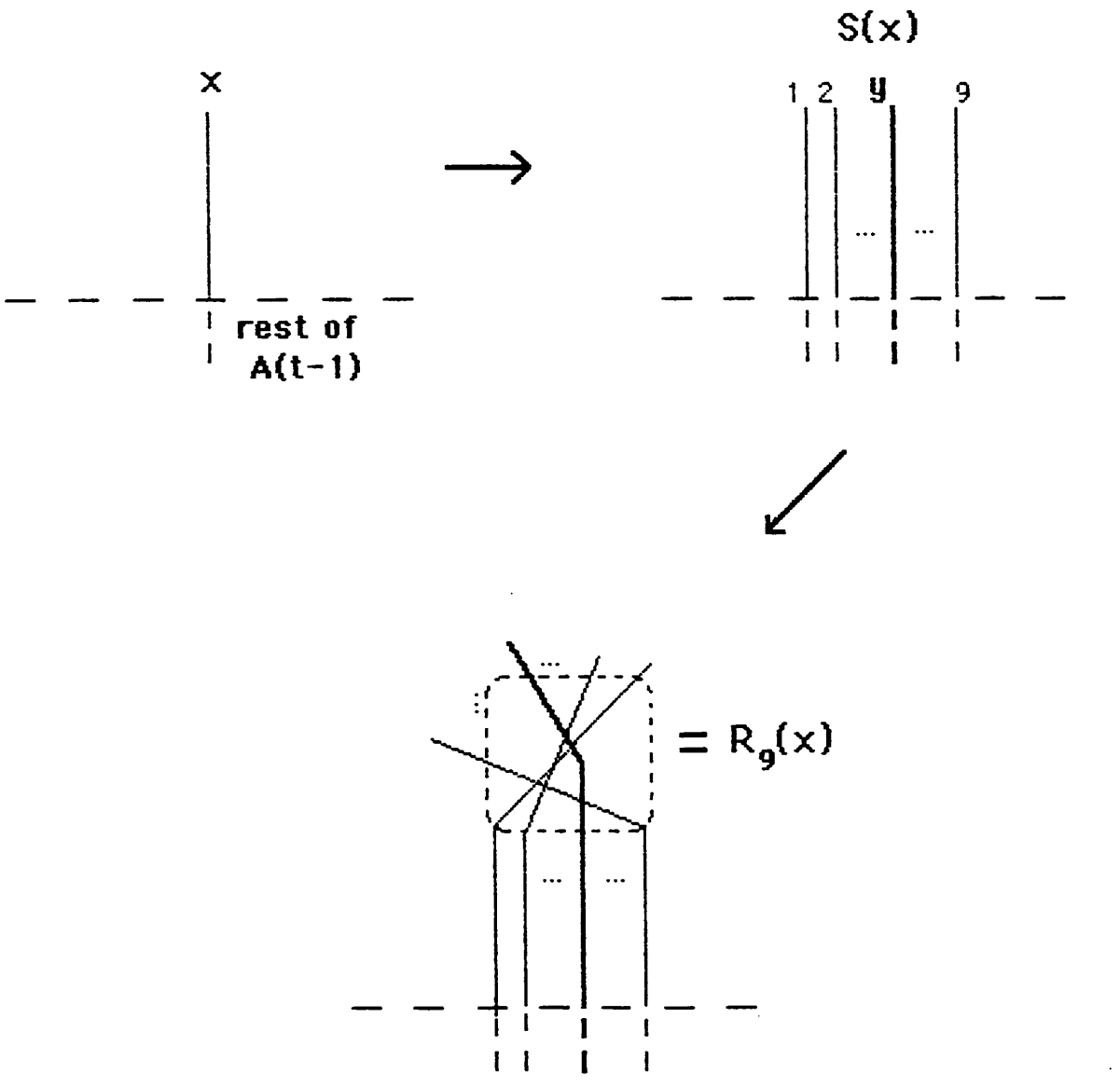
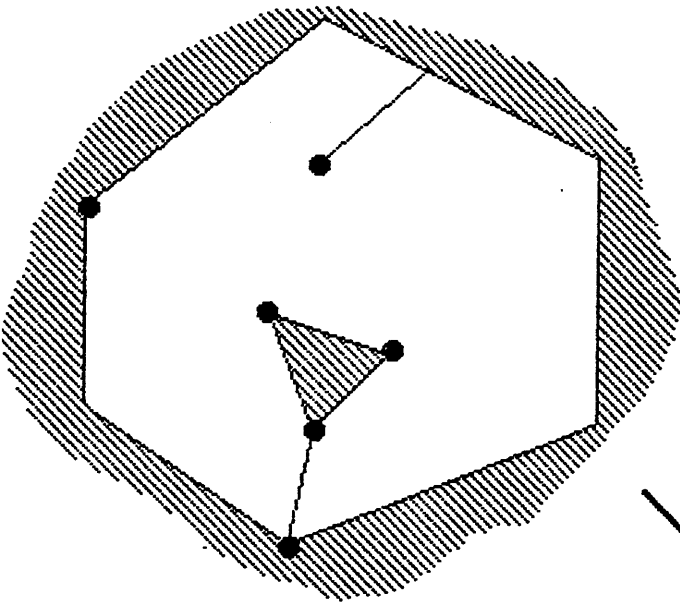
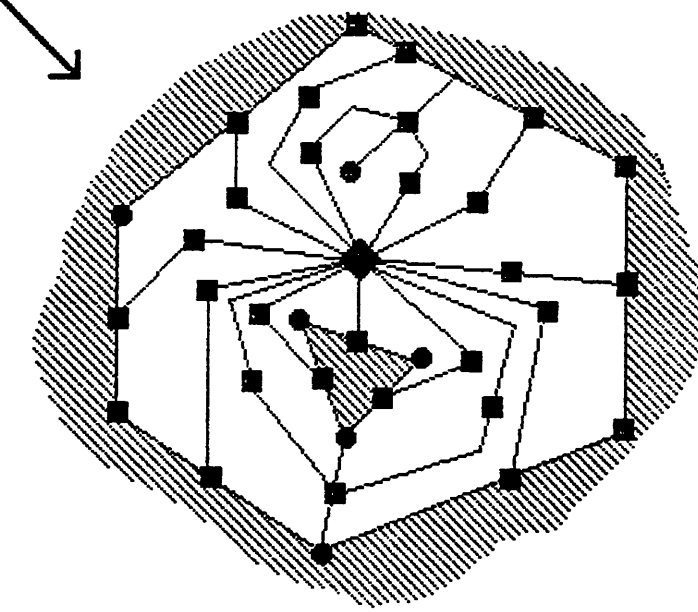


FIGURE 2

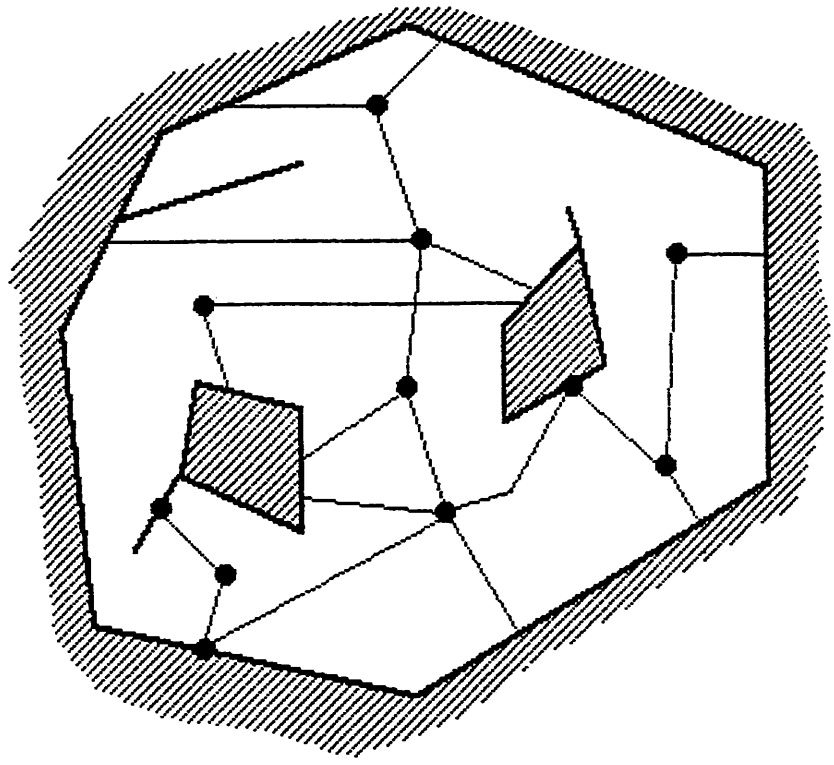


A face F of a piece
vertices of G denoted by ●



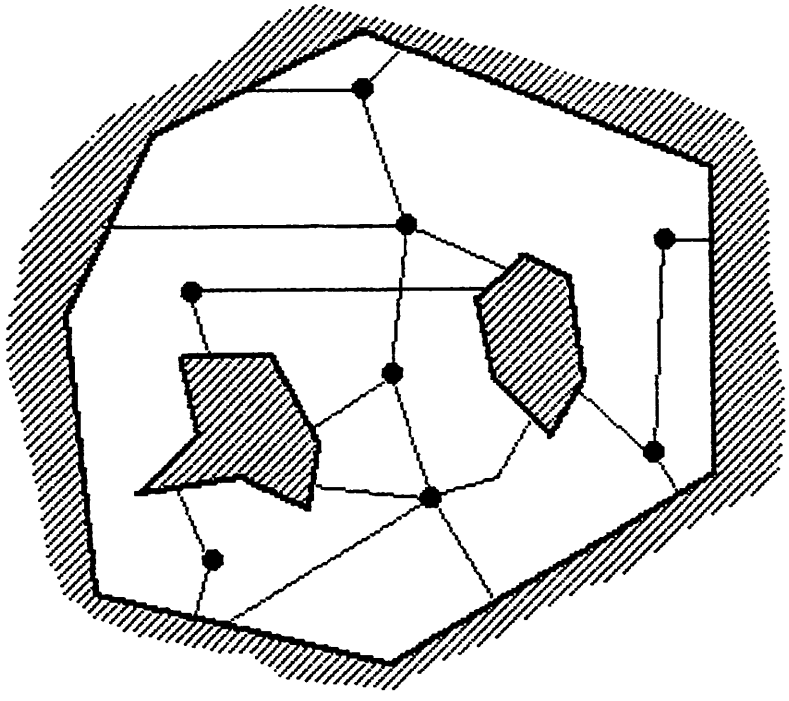
w(F) indicated by ◆
other new vertices indicated by ■

FIGURE 3



(a)

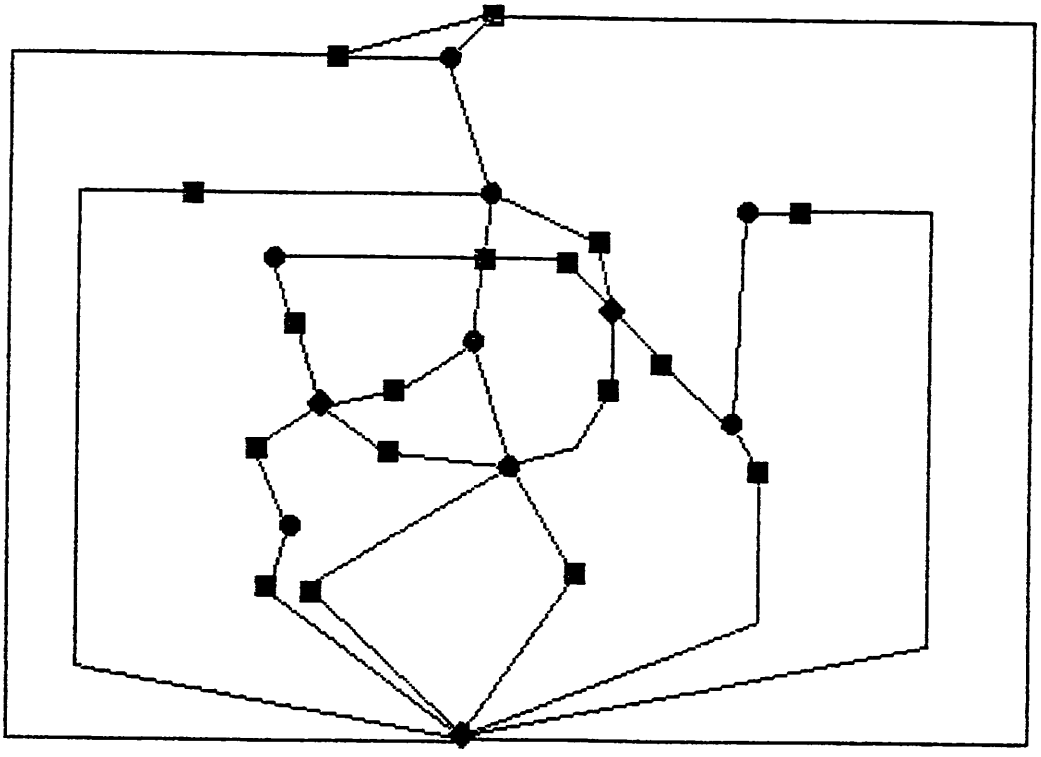
A plot that encloses two pieces and is enclosed by one.



(b)

After expanding the boundaries.

FIGURE 4

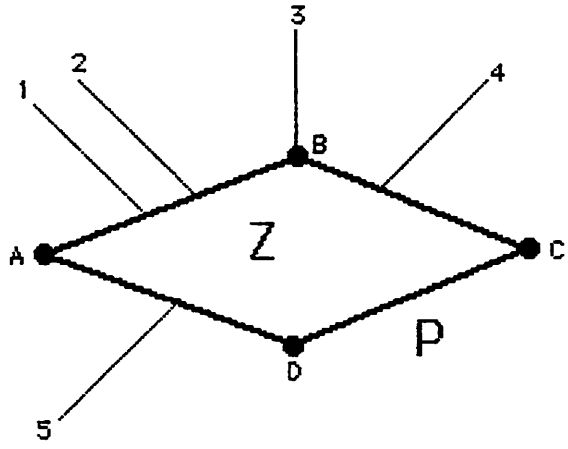


(C) The resulting planar graph

vertices of type v(z) indicated by ◆

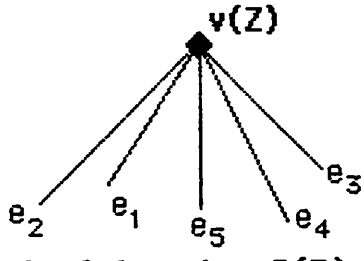
other added vertices indicated by ■

FIGURE 4 (CONT'D)

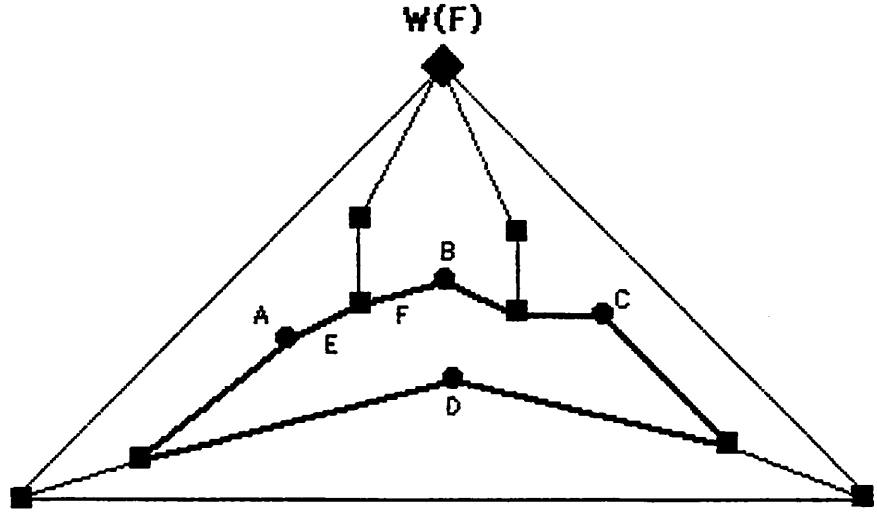


(a) Original drawing of Z and P

- only the boundary sections of Z shown, in bold
- only the linking sections of Z and P shown, in light

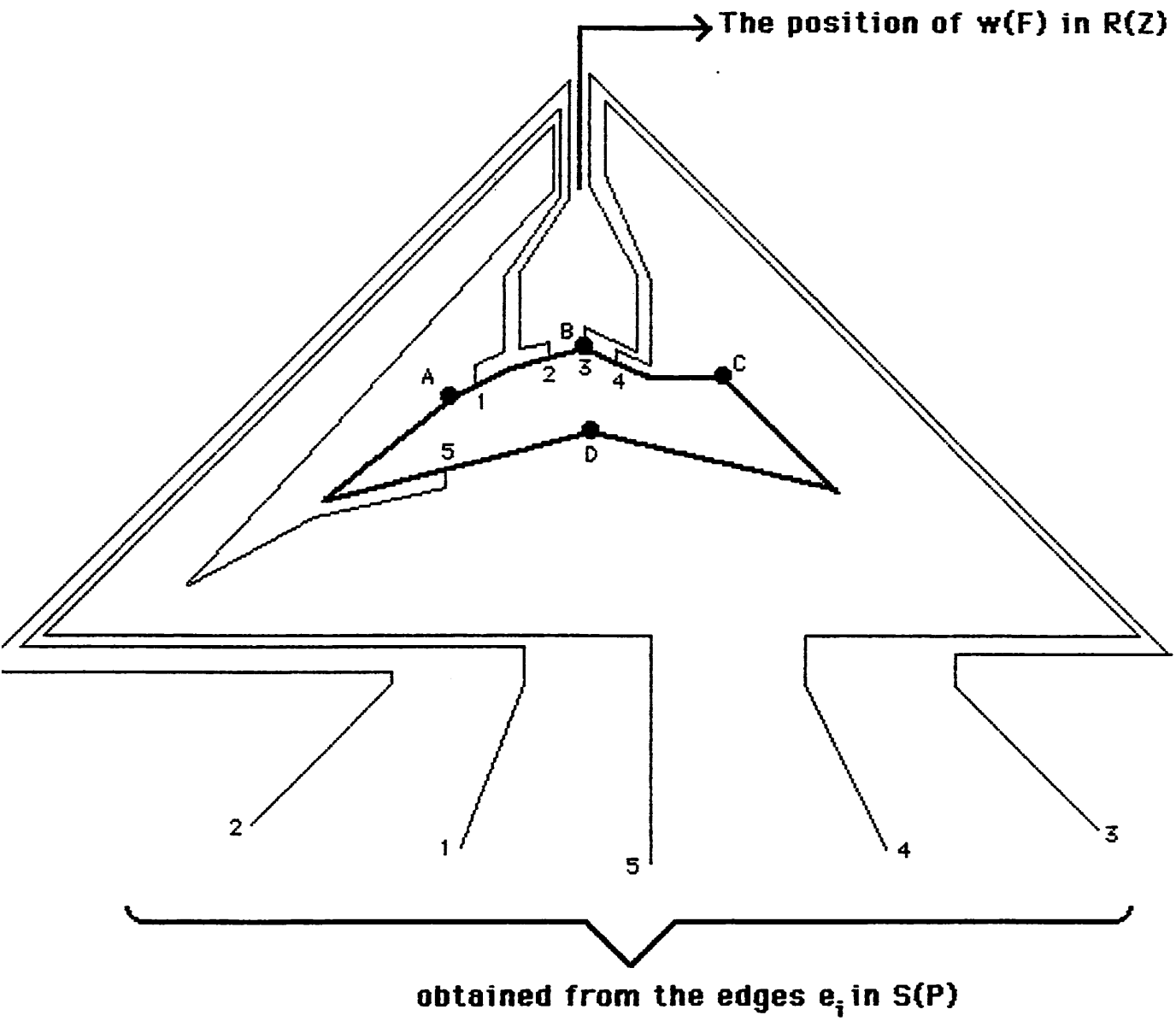


(b) Part of drawing S(P)



(c) part of the drawing R(Z)

FIGURE 5 (a,b,c)



(d) Integrating the drawings of Z and P

FIGURE 5 (d)

Polyhedral Results for the Precedence-Constrained Knapsack Problem

E. Andrew Boyd
Department of Mathematical Sciences
Rice University

Abstract

A problem characteristic common to a number of important integer programming problems is that of precedence constraints: a transitive collection of constraints of the form $x_j \leq x_i$ with $0 \leq x_i \leq 1$, $0 \leq x_j \leq 1$, x_i, x_j integer. Precedence constraints are of interest both because they arise frequently in integer programming applications and because the convex hull of feasible integer points is the same as the region obtained by relaxing the integrality restrictions. This paper investigates the polyhedral structure of the convex hull of feasible integer points when the precedence constraints are complicated by an additional constraint or, more generally, by additional constraints defining an independence system. A general lifting theorem for independence systems with precedence constraints is presented, and extensions of the cover and 1-configuration inequalities known for the knapsack polytope are discussed. A general procedure for inducing facets called rooting is introduced, and a variety of facets based on rooting are presented. The paper concludes by discussing a procedure for coalescing constraints related to minimal covers into facets.

1 Introduction

A problem characteristic common to a number of important integer programming problems is that of precedence constraints: a transitive collection of constraints of the form $x_j \leq x_i$ where $0 \leq x_i \leq 1$, $0 \leq x_j \leq 1$, x_i, x_j integer. Such constraints are used to model logical precedence conditions such as "retail outlet j cannot be stocked from warehouse i unless warehouse i is built," or logical forcing conditions such as "if the generator is on in period j it must also be on in period i." Beyond the extensive practical value of precedence constraints for modeling, precedence constraints are of theoretical and computational interest because by themselves they define a polyhedron with integer vertices. Integer programs defined completely by precedence constraints can therefore be solved by relaxing the integrality restrictions and solving the resulting linear program.

More formally, let P^{LP} be the polyhedron obtained by relaxing the integrality restrictions of an integer program and let P be the convex hull of feasible integer points for the same problem. It is always true that $P \subseteq P^{LP}$, and the observation made above is that an integer program defined

entirely by precedence constraints has the very special property $P = P^{LP}$. The purpose of the present work is to explore the polyhedral structure of P when it is defined not only by a collection of precedence constraints but is complicated by an additional constraint or constraints. It is well-known that optimizing a linear function on P when it is defined by the constraints $0 \leq x_i \leq 1$ together with an arbitrary less-than-or-equal-to constraint with non-negative coefficients — the polyhedral version of the so-called *knapsack problem* — is already NP-complete. As the knapsack problem is nothing more than a special case of the *precedence-constrained knapsack problem* it follows that this latter problem is also NP-complete.

While the simple knapsack problem is known to be solvable in pseudopolynomial time using dynamic programming, there is no obvious similar formulation for the precedence-constrained knapsack problem. Polyhedral solution procedures may thus prove to be the most efficient methods for solving this problem. As there are a host of important problems that can be formulated as precedence-constrained knapsack problems — for example, capital budgeting under precedence constraints — the present work has direct practical consequences. Further, it is envisioned that the present work can be used effectively in the solution of general integer programming problems in much the same way as polyhedral results for the simple knapsack problem were used by Crowder, Johnson, and Padberg in their Lanchester prize-winning paper [6].

While motivation for studying the precedence-constrained knapsack problem was initially provided by the properties of precedence constraints, some of the present work can be interpreted as extending polyhedral results for the simple knapsack problem (see, for example, [1], [2], [17], and [20]). In many ways the results for the simple knapsack problem extend quite naturally to the more general case. Yet, as many of the results presented here attest to, the additional structure of the precedence-constrained problem is sufficiently rich that results for the simple knapsack problem do not fully capture it.

The following section provides necessary background material and notation for the results presented in the remaining sections. It is assumed that the reader is familiar with basic concepts related to polyhedral theory. Section 3 investigates conditions under which many of the constraints in a natural integer programming formulation of the precedence-constrained knapsack problem are facets of P. In Section 4 it is demonstrated that the well-known sequential lifting procedure for the simple knapsack problem can be generalized to the case of the precedence-constrained knapsack problem.

Section 5 introduces two classes of facets that are extensions of well-known facets for the simple knapsack problem. In Section 6, a general procedure for inducing facets called *rooting* is introduced, and this technique is used in Sections 6 and 7 to show how facets can be constructed from valid inequalities that are not facets. Section 8 presents facets coalesced from a collection of invalid inequalities related to minimal cover inequalities.

2 Background and Notation

A *partially ordered set* $(V, \preceq)$ is a collection of elements V together with a binary relation $\preceq$ that is reflexive, antisymmetric, and transitive. Using the obvious notation, $i \prec j$ will mean $i \preceq j$ and $i \neq j$. An element j is said to *cover* i if $i \prec j$ and there exists no element k such that $i \prec k \prec j$. Given a set $S \subseteq V$, we denote by $G(S)$ the set of elements $i \in S$ for which there are no $j \in S$ such that $j \prec i$. Likewise, we denote by $H(S)$ the set of elements $i \in S$ for which there are no $j \in S$ such that $i \prec j$.

A Hasse diagram is a directed graph (V, E) with an edge i-$j \in E$ if and only if j covers i. In order to emphasize this interpretation of the partial order the elements in V generally will be referred to as vertices. In keeping with commonly accepted practice, when drawing a Hasse diagram edge direction will be implied by the relative vertical location of two vertices in the diagram — j covers i in such a diagram if there is an edge between i and j and j is located above i in the diagram. A *lower ideal*

is a set $S \subseteq V$ such that if $j \in S$ and $i \preceq j$ then $i \in S$. Similarly, the lower ideal generated by a set S, denoted $\ell(S)$, is the set of i such that $i \preceq j$ for some $j \in S$. For singleton sets $\{i\}$ we write $\ell(i)$ rather than $\ell(\{i\})$. The set of all lower ideals of a partially ordered set $(V, \preceq)$ will be denoted by $\mathcal{L}$ with $\mathcal{L}_S$ denoting the set of all lower ideals of the partially ordered set $(S, \preceq)$, $S \subseteq V$. An *upper ideal* is a set $S \subseteq V$ such that if $j \in S$ and $j \preceq i$ then $i \in S$. The upper ideal generated by a set S will be denoted $u(S)$ with $u(i)$ denoting the upper ideal generated by a singleton set $\{i\}$.

Given a set of elements V, let x_i be a real-valued variable associated with element $i \in V$. For any set $S \subseteq V$, $\mathbf{R}^S$ will denote the $|S|$-dimensional space associated with the variables x_i, $i \in S$. The notation x^S will be used for the incidence vector of S, namely, $x_i = 1$ if $i \in S$ and $x_i = 0$ if $i \notin S$. Given an $m \times |V|$ matrix a, the notation $a(S)$ will denote $\sum_{i \in S} a_i$, where a_i represents column i of a. Given a vector $b \in \mathbf{R}^m$, if every component of $a(S)$ is less than or equal to every component of b we write $a(S) \leq b$, whereas if $a(S)$ exceeds b in at least one component we write $a(S) > b$. As in the notation for the lower ideal of a set we write $a(i)$ rather than $a(\{i\})$ for singleton sets.

3 The Problem

Formally, the problem of interest can be stated as follows.

The Precedence-Constrained Knapsack Problem (PK):
Given a partially ordered set $(V, \preceq)$ and functions $w : V \to \mathbf{R}$, $a : V \to \mathbf{R}^+$, find an $S \in \mathcal{L}$ satisfying $a(S) \leq \alpha$ that maximizes $w(S)$.

Note that the coefficients a_i are restricted to be non-negative so that given a set $S \subseteq V$ satisfying $a(S) \leq \alpha$, any set $T \subseteq S$ satisfies $a(T) \leq \alpha$; that is, the set of $S \subseteq V$ satisfying $a(S) \leq \alpha$ constitutes an independence system on V. Indeed, the results of the following development are true more generally for the case of precedence constraints imposed on an independence system. Specifically, if a is interpreted as a non-negative $m \times |V|$ matrix and α an m-vector, all of the following results are true as stated.

It will often prove useful to assume the elements of V or of some subset of V are indexed so that they satisfy the following property.

Property 1 *If $i \preceq j$ then $i \leq j$.*

Clearly there always exists such an indexing and in general many such indexings exist. Note that if $a(\ell(i)) > \alpha$ for some $i \in V$ then clearly i cannot be in any feasible solution to PK. Further, it is easy to determine if a vertex satisfies this inequality. We therefore assume henceforth that all $i \in V$ satisfy $a(\ell(i)) \leq \alpha$.

Associating a variable x_i with each $i \in V$, a valid integer programming formulation of PK is the following.

$$
\begin{array}{lll}
\max & \sum_{i=1}^{|V|} w_i x_i & \\
\text{s.t.} & x_i \geq 0 & i \in H(V) \qquad (1) \\
& x_i \leq 1 & i \in G(V) \qquad (2) \\
& x_j \leq x_i & j \text{ covers } i \qquad (3) \\
& \sum_{i=1}^{|V|} a_i x_i \leq \alpha & \qquad (4) \\
& x_i \text{ integer} & i \in V \qquad (5)
\end{array}
$$

Let P^{LP} denote the polyhedron defined by (1), (2), (3), and (4) and let $P = \text{conv}(\{x^S : S \in \mathcal{L},\ a(S) \leq \alpha\})$. Further, given a set $S \subseteq V$, let $P_S = \text{conv}(\{x^{T \cap S} : T \in \mathcal{L},\ a(T) \leq \alpha\})$. Note that for $S \in \mathcal{L}$, $P_S = P \cap \mathbf{R}^S$.

As it is trivial to construct objective functions for which an integral optimal solution does not exist, it follows that $P \neq P^{LP}$. However, while this implies that (1), (2), (3), and (4) do not constitute all of the facets of P, many are often facets. We first make note of the dimension of P before considering conditions under which (1), (2), and (3) are facets of P.

Proposition 1 *P has dimension $|V|$.*

Proof: Assume V is indexed so that Property 1 is satisfied. Let M be a matrix with rows indexed $1, \ldots, |V|$ and let column i be the vector $x^{\ell(i)}$. By Property 1 the matrix M is upper triangular with nonzeros on the diagonal. It follows that the columns of M together with the 0 vector form a set of $|V| + 1$ affinely independent points contained in P. □

Proposition 2 *The constraints (1) are facets of P.*

Proof: Consider any $i \in H(V)$ and assume V is indexed so that Property 1 is satisfied and that in addition $i = |V|$. Letting M be the matrix described in Proposition 1, it is clear that the first $|V| - 1$ columns of M together with the 0 vector form a set of $|V|$ affinely independent points satisfying $x_i = 0$. □

Proposition 3 *A constraint of the form (2) is a facet of P if and only if*

$$a(\ell(j) \cup \{i\}) \leq \alpha$$

for all $j \in V$.

Proof: Assume V is indexed so that Property 1 is satisfied and that in addition $i = 1$. Let M be a matrix with rows indexed $1, \ldots, |V|$ and let column j be the vector $x^{\ell(j) \cup \{i\}}$. By assumption each vector $x^{\ell(j) \cup \{i\}} \in P$ and clearly $x_i^{\ell(j) \cup \{i\}} = 1$. By Property 1, M is upper triangular with nonzeros on the diagonal. Affine independence of the columns of M follows, completing half of the proof.

To complete the other half of the proof, assume there exists some $j \in V$ such that $a(\ell(j) \cup \{i\}) > \alpha$. This implies that any point $x \in P$ satisfying $x_i = 1$ must also satisfy $x_j = 0$. The constraint $x_i \leq 1$ therefore intersects P in dimension at most $|V| - 2$ and thus cannot be a facet. □

Proposition 4 *A constraint of the form (3) is a facet of P if and only if*

$$a(\ell(k) \cup \ell(j)) \leq \alpha$$

for all k with $i \in \ell(k)$.

Proof: Consider any two $i, j \in V$ such that j covers i and let $I_0 = \{k \in V : i \notin \ell(k)\}$ and $I_1 = \{k \in V : i \in \ell(k)\}$. Assume that V is indexed so that Property 1 is satisfied and that in addition $p < q$ for $p \in I_0$, $q \in I_1$. These assumptions imply $i = |I_0| + 1$. Further, j can be indexed $|I_0| + 2$ without violating these assumptions. If not, there would exist some $k \in I_1$ other than i and j such that $k \in \ell(j)$, which would contradict the assumption that j covers i. Let M be a matrix with rows indexed $1, \ldots, |V|$ and let column k be the vector $x^{\ell(k)}$ for $k = 1, \ldots, |I_0| + 2$ and $x^{\ell(k) \cup \ell(j)}$ otherwise. By assumption, all of the columns of M are contained in P and by Property 1 M is upper triangular with nonzeros on the diagonal. Further, columns $k \leq |I_0|$ satisfy $x_i = x_j = 0$ while columns $k \geq |I_0| + 2$ satisfy $x_i = x_j = 1$. The columns of M other than column $|I_0| + 1$ together

Input: A facet $\sum_{i \in B} d_i x_i \leq \beta$ of P_B where $B \in \mathcal{L}$.

Note: It is assumed that V is indexed so that Property 1 is satisfied and that in addition $p < q$ for $p \in B$, $q \in V \backslash B$.

Output: A facet $\sum_{i \in V} b_i x_i \leq \beta$ of P.

begin

let $b_i = d_i$ for $i \in B$, $b_i = 0$ otherwise

for $i = |B| + 1, \ldots, |V|$ do

$$b_i = \beta - \max_{\{S \in \mathcal{L}:\, i \in S,\, a(S) \leq \alpha\}} b(S)$$

end

Algorithm 1

with the 0 vector form a set of $|V|$ affinely independent points satisfying $x_i = x_j$, completing half of the proof.

To complete the proof, assume there exists some k with $i \in \ell(k)$ such that $a(\ell(k) \cup \ell(j)) > \alpha$. This implies that any point $x \in P$ satisfying $x_k = 1$ has $x_i = 1$ and $x_j = 0$; that is, any point $x \in P$ satisfying $x_i = x_j$ satisfies $x_k = 0$. The constraint $x_j \leq x_i$ therefore intersects P in dimension at most $|V| - 2$ and thus cannot be a facet. □

4 Liftings

Unlike the valid inequalities discussed in the previous section the valid inequalities discussed in the remaining sections are not generally facets of P but are instead facets of a lower-dimensional polyhedron P_S with $S \in \mathcal{L}$. Similar results arise in the study of the polyhedral structure of many problems. For the simple knapsack problem Padberg [16] showed how facets of knapsack problems defined on subsets of V could be algorithmically lifted to facets of the full knapsack problem on V. Padberg's result was an instance of a more general result proved by Nemhauser and Trotter [13] related to polyhedra associated with independence systems.

In this section we present a procedure for lifting facets of P_S with $S \in \mathcal{L}$ to facets of P. The development is very much in the spirit of the results mentioned above. In fact, the contribution of the present lifting theorem is the recognition that a facet of P is generated if the sequential lifting procedure respects the underlying partial order on V.

Theorem 1 *The constraint generated by Algorithm 1 is a facet of P.*

Proof: Clearly, $\sum_{i \in V} b_i x_i \leq \beta$ is valid at the beginning of Algorithm 1 by choice. As b_i is chosen at each iteration so as to maintain feasibility, the constraint generated by Algorithm 1 is valid at the conclusion of the algorithm.

To prove that the constraint is a facet of P it remains to provide $|V|$ affinely independent points in P satisfying this constraint at equality. To this end, let $L(i) \in \mathcal{L}$ be some set achieving the optimal value in the maximization problem of Algorithm 1 at iteration i, noting that $i \in L(i)$. Since $b_j = 0$ for $j > i$ at iteration i, the only reason $L(i)$ might need to contain some vertex $j > i$ would

be that $j \in \ell(k)$ for some $k \in \{1,\ldots,i\}$. However, by Property 1 this cannot be the case. Thus, we can further assume $L(i) \subseteq \{1,\ldots,i\}$.

By the choice of the value of b_i at iteration i of Algorithm 1, it follows that $b(L(i)) = \beta$ for all iterations after iteration i, and in particular that $b(L(i)) = \beta$ for $i = |B|+1,\ldots,|V|$ at the termination of Algorithm 1. As $\sum_{i\in B} d_i x_i \leq \beta$ is a facet of P_B, there exists some collection of $|B|$ sets $B(i) \in \mathcal{L}_B$ with $a(B(i)) \leq \alpha$ such that the vectors $x^{B(i)}$ are affinely independent and $b(B(i)) = \beta$. Thus, to complete the proof it remains only to show that the vectors $x^{B(i)}$ and $x^{L(i)}$ are affinely independent.

Consider the matrix M_0 constructed as follows. Let row i correspond to vertex $i \in V$. Similarly, let columns $i = 1,\ldots,|B|$ be $x^{B(i)}$ and let columns $i = |B|+1,\ldots,|V|$ be $x^{L(i)}$. Let M_1 be the $|V| \times |V|-1$ matrix formed by subtracting column 1 from each column in M_0 and deleting column 1.

Since $B(i) \subseteq B$, the $|V\backslash B| \times |B|-1$ lower-left submatrix of M_1 is the zero matrix. Further, since $L(i)$ was chosen so that $i \in L(i)$ and $L(i) \subseteq \{1,\ldots,i\}$ the $|V\backslash B| \times |V\backslash B|$ lower-right submatrix of M_1 is upper triangular with nonzero elements on the diagonal. It follows that any vector y satisfying $M_1 y = 0$ must have its last $|V\backslash B|$ elements equal to 0. As the remaining $|B|-1$ columns corresponding to the vectors $x^{B(i)}$ are linearly independent the proof is complete. □

The facet generated by Algorithm 1 will generally be affected by the indexing of the set $V\backslash B$. All that is required of the indexing is that it must satisfy Property 1, and different choices for the indices will generally lead to different facets.

It is also interesting to note that the maximization problems solved in Algorithm 1 are instances of the problem PK itself. The difficulty of actually solving these problems is reduced in practice by recognizing that at iteration i the maximization problem can be solved on $\mathcal{L}_{\{1,\ldots,i\}}$. Further, these problems are generated in a special fashion and are not indicative of the most general problems PK. An interesting open question is whether an efficient algorithm exists for solving the maximization problems encountered in Algorithm 1 when lifting the facets introduced in the following section.

5 Two Classes of Facets

The present section develops two classes of inequalities that can be lifted into facets of P using the results of the previous section. Following the development of polyhedral results for the knapsack problem, we define a *cover* as any set $C \in \mathcal{L}$ such that $a(C) > \alpha$. If for any set $S \subseteq H(C)$ with $|S| = K$ it is true that $a(\ell(S)) > \alpha$ but for any $i \in S$, $a(\ell(S)\backslash\{i\}) \leq \alpha$ then C is called a *K-cover*. The following theorem characterizes when K-covers define facets of P_C.

Theorem 2 *Given any K-cover C the constraint*

$$x(H(C)) \leq K-1 \tag{6}$$

is a facet of P_C if and only if

$$\bigcap_{\{S\subseteq H(C):\ |S|=K-1\}} \ell(S) = \phi\ .$$

Proof: Clearly, (6) is valid for P_C by definition. Suppose $\bigcap_{\{S\subseteq H(C):\ |S|=K-1\}} \ell(S) \neq \phi$, and let i be some element in this set. Since any set $T \subseteq C$ satisfying $x^T \in P_C$ and $x^T(H(C)) = K-1$ must contain i, it follows that all such points satisfy $x_i^T = 1$. As (6) is not a scalar multiple of the constraint $x_i \leq 1$ it cannot be a facet of P_C.

To see that (6) is a facet of P_C when $\bigcap_{\{S\subseteq H(C):\ |S|=K-1\}} \ell(S) = \phi$, let

$$\beta(C) \leq \omega \tag{7}$$

be such that $\beta(T) = \omega$ for all sets $T \subseteq C$ with $x^T \in P_C$ satisfying (6) at equality. We will show that (7) is a scalar multiple of (6), thus establishing that (6) is a facet of P_C.

Consider first the case of an element $i \in C \backslash H(C)$. Since $\bigcap_{\{S \subseteq H(C):\ |S|=K-1\}} \ell(S) = \phi$, there exists some set $T \subseteq H(C)$ with $|T| = K-1$ such that $i \notin \ell(T)$ and some $j \in H(C) \backslash T$ such that $i \in \ell(j)$. By the definition of a K-cover, $a(\ell(T) \cup \ell(j) \backslash \{j\}) \leq \alpha$ and thus since $\ell(i) \subseteq \ell(j) \backslash \{j\}$, $a(\ell(T) \cup \ell(i)) \leq \alpha$. Thus, $x^{\ell(T) \cup \ell(i)}$ and $x^{\ell(T) \cup \ell(i) \backslash \{i\}}$ are both contained in P_C and satisfy (7) at equality. It follows that $\beta_i = 0$.

Thus, consider the case of an element $i \in H(C)$. Let $S \subseteq H(C)$ with $i \notin S$ and $|S| = K-1$. Further, let $j \in S$. By the definition of K-cover, $a(\ell(i) \cup \ell(S) \backslash \{j\}) \leq \alpha$ and $a(\ell(i) \backslash \{i\} \cup \ell(S)) \leq \alpha$. Thus, $x^{\ell(i) \cup \ell(S) \backslash \{j\}}$ and $x^{\ell(i) \backslash \{i\} \cup \ell(S)}$ are both contained in P_C and satisfy (7) at equality. As the sets $\ell(i) \cup \ell(S) \backslash \{j\}$ and $\ell(i) \backslash \{i\} \cup \ell(S)$ differ by a single element, it follows that $\beta_i = \beta_j$.

We are thus able to conclude that (7) must be a scalar multiple of (6). □

The following corollary of Theorem 2 will prove useful for future reference.

Corollary 1 *Let $C \in \mathcal{L}$, let $1 \leq J \leq |H(C)| - 1$, and assume that $\bigcap_{\{S \subseteq H(C):\ |S|=J-1\}} \ell(S) = \phi$. Then there exist $|C|$ affinely independent vectors x^Q such that $Q \in \{T \subseteq \mathcal{L}_C :\ x^T(H(C)) = J-1$ and $T \subseteq \ell(S)$ where $S \subseteq H(C)$ with $|S| = J\}$.*

An important special case of a K-cover C arises when $K = |H(C)|$. As there exists no set $S \in \mathcal{L}_C$ other than $S = C$ that is a cover, a $|H(C)|$-cover is also called a *minimal* cover. More important, however, is that every K-cover can be generated from some minimal cover by the lifting procedure described in the previous section. General K-covers are thus simply instances of lifted minimal covers.

For minimal covers it is easily verified that the condition $\bigcap_{\{S \subseteq H(C):\ |S|=|H(C)|-1\}} \ell(S) = \phi$ is equivalent to $\ell(i) \cap \ell(j) = \phi$ for all $i, j \in H(C)$. The following theorem shows the strength of minimal cover constraints when they are facets.

Theorem 3 *If the knapsack constraint (4) is of the form*

$$x(H(V)) \leq |H(V)| - 1 \tag{8}$$

and is a facet of P, that is, $\ell(i) \cap \ell(j) = \phi$ for all $i, j \in H(V)$, then the set of constraints (1), (2), (3), and (8) provide a complete description of P.

Proof: Denote the set of vertices in $H(V)$ by $v(1), \ldots, v(|H(V)|)$. Let M be the constraint matrix defined by (1), (2), (3), and (8) after multiplying the constraints (1) by -1 so that the problem is defined completely by $\leq$ constraints. Since $\ell(i) \cap \ell(j) = \phi$ for all $i, j \in H(V)$, M is block diagonal with a single complicating constraint, specifically, constraint (8). Let $N_1, \ldots, N_{|H(V)|}$ denote the block matrices along the diagonal with matrix N_i corresponding to the constraints associated with vertices in $\ell(v(i))$.

Consider the dual of the linear program formed by optimizing an arbitrary objective function subject to the constraints defined by M. The constraint matrix M^T of the dual is very nearly the constraint matrix of $|H(V)|$ disjoint network flow problems. The complications are that each matrix N_i^T has a column with a single -1 corresponding to the constraint (1) associated with vertex $v(i)$ and a collection of columns with a single 1 corresponding to constraints (2). In addition, there is a complicating column in M^T corresponding to constraint (8).

These complications can be alleviated as follows. Note that if the rows of the matrix N_i^T are summed, the resultant vector has the property that it has a -1 in the entry corresponding to the constraint (1), a 1 in entries corresponding to constraints (2), and a 0 in entries corresponding to constraints (3). Thus, summing the rows of M^T that contain rows of $N_{|H(V)|}^T$ and appending the

negative of the resultant vector to M^T yields a matrix in which the columns that contain columns of $N^T_{|H(V)|}$ now have a single 1 and a single -1. In addition, the new row of M^T has a -1 in the column corresponding to constraint (8). Similarly, summing the rows of M^T that contain rows of N^T_i for some fixed $i < |H(V)|$ and subtracting the result from the row corresponding to the vertex $v(i+1)$ has the following effect. All columns of M^T that contain columns of N^T_i have a single 1 and a single -1. Further, the 1 in the column corresponding to constraint (8) in row $v(i+1)$ is eliminated. Performing this row operation in the order $i = |H(V)| - 1, \ldots, 1$, the resultant matrix has a single 1 and a single -1 in each and every column.

As the transformed matrix M^T is a network flow matrix, it follows that this dual problem always has an integer optimal solution if it is bounded and its right-hand-side is integral. Thus the original system of constraints defined by M is totally dual integral and it follows that all of the extreme points of P are integral. □

While Theorem 3 demonstrates the extent of the strength of minimal cover constraints when they are facets, the equivalent condition that $\ell(i) \cap \ell(j) = \phi$ for all $i, j \in H(C)$ is restrictive. Often, however, the process of lifting a constraint using the procedure outlined in the previous section can generate a facet of a higher dimensional polytope even if the minimal cover does not satisfy this condition. For example, it is easy to construct K-covers C such that $H(C)$ satisfies the conditions of Theorem 3 but such that every minimal cover $\ell(T)$ defined by $T \subseteq H(C)$ with $|T| = K$ does not satisfy the conditions of this theorem. Alternatively, it is possible to consider strengthening constraints (6) associated with minimal covers that do not satisfy the conditions of Theorem 2. Such methods are developed in the following sections.

A related class of inequalities is a generalization of the class of 1-configurations introduced by Padberg [17]. Let $D \in \mathcal{L}$ be a cover such that for some fixed $k \in H(D)$, $a(D\backslash\{k\}) \leq \alpha$. If for any set $S \subseteq H(D)\backslash\{k\}$ with $|S| = J \geq 2$ it is true that $a(\ell(k) \cup \ell(S)) \geq \alpha$ but for any $i \in S$, $a(\ell(k) \cup \ell(S)\backslash\{i\}) \leq \alpha$ then D is called a *1-configuration*. Note that if $J = |H(D)| - 1$ then a 1-configuration is simply a minimal cover.

Theorem 4 *Let D be a 1-configuration and let $H_K(D)$ denote any subset of $H(D)\backslash\{k\}$ of cardinality K. For any $J \leq K \leq |H(D)| - 1$ the constraint*

$$(K - J + 1)x_k + x(H_K(D)) \leq K \tag{9}$$

is a facet of P_D if and only if

$$\ell(k) \cap \ell(H_K(D)) = \phi \quad \textit{and} \tag{10}$$

$$\bigcap_{\{S \subseteq H_K(D):\, |S| = J-1\}} \ell(S) = \phi\,. \tag{11}$$

Proof: The validity of constraints of the form (9) follows from the definition of a 1-configuration.

Suppose $\ell(k) \cap \ell(H_K(D)) \neq \phi$ and let i be some element in this set. Let $T \subseteq D$ be such that $x^T \in P_D$ and x^T satisfies (9) at equality. If $k \in T$ then $i \in T$ since $i \in \ell(k)$. If $k \notin T$, then $\ell(H_K(D)) \subseteq T$ in order for x^T to satisfy (9) at equality, but again this implies $i \in T$. Thus, any $x^T \in P_D$ satisfying (9) at equality also satisfies $x^T_i = 1$, and as (9) is not a scalar multiple of $x_i \leq 1$ it cannot be facet of P_D.

Thus, suppose $\bigcap_{\{S \subseteq H_K(D):\, |S| = J-1\}} \ell(S) \neq \phi$ and let i be some element in this set. Clearly, any set $T \subseteq D$ such that $x^T \in P_D$ satisfying (9) at equality must contain a set $S \subseteq H_K(D)$ with $|S| = J - 1$. Since $\ell(S) \subseteq T$ it follows that $i \in T$. However, once again it has been shown that any $x^T \in P_D$ satisfying (9) at equality also satisfies $x^T_i = 1$, implying (9) cannot be a facet of P_D.

To prove that any constraint of the form (9) is a facet of P_D under the stated conditions we demonstrate that it is a facet of $P_{\ell(k) \cup \ell(H_K(D))}$. Since it is easily verified that the properties of a 1-configuration are such that lifting a constraint (9) from $P_{\ell(K) \cup (H_K(D))}$ to P_D does not change the constraint, it follows that any constraint (9) is in fact a facet of P_D.

Let $\ell(k) \cup \ell(H_K(D))$ be indexed so as to satisfy Property 1 with the additional condition that if $i \in \ell(k)$ and $j \in \ell(H_K(D))$ then $i < j$; that is, the first $|\ell(k)|$ indices belong to the vertices in $\ell(k)$.

Construct an $|\ell(k) \cup \ell(H_K(D))| \times |\ell(k) \cup \ell(H_K(D))|$ matrix M_0 as follows. Let column 1 be the vector $x^{\ell(H_K(D))}$ and let columns $i = 2, \ldots, |\ell(k)|$ be $x^{\{1,\ldots,i-1\} \cup \ell(H_K(D))}$. Let the remaining $|\ell(H_K(D))|$ columns be $x^{\ell(k) \cup Q(i)}$, where $Q(i) \in \mathcal{L}_{\ell(H_K(D))}$, $x^{Q(i)}(H_K(D)) = J - 1$, and the vectors $x^{Q(i)}$ are affinely independent. The existence of such a collection of vectors $x^{Q(i)}$ follows from Corollary 1. Note that the columns of M_0 are contained in $P_{\ell(k) \cup \ell(H_K(D))}$ and satisfy (9) at equality.

Let M_1 be the $|\ell(k) \cup \ell(H_K(D))| \times |\ell(k) \cup \ell(H_K(D))| - 1$ matrix obtained from M_0 by subtracting column $|\ell(k) \cup \ell(H_K(D))|$ from all other columns. The resultant $|\ell(k)| \times |\ell(k)|$ upper-left submatrix of M_1 is lower triangular with nonzero entries on the diagonal while the upper-right $|\ell(k)| \times |\ell(H_K(D))| - 1$ submatrix is the 0 matrix. It follows that any vector y satisfying $M_1 y = 0$ has $y_i = 0$ for $i = 1, \ldots, |\ell(k)|$. As the remaining columns of M_1 are linearly independent, it follows that the columns of M_0 are affinely independent. □

It is not difficult to verify that except for the case where $K = J$, constraints of the form (9) cannot arise as liftings of minimal cover inequalities so that 1-configurations are indeed fundamentally different than cover inequalities.

6 Rooting and Zigzags

Consider a minimal cover C and the associated minimal cover inequality (6). As was proved in the previous section, $\ell(i) \cap \ell(j) = \phi$ for all $i, j \in H(C)$ is a necessary condition for (6) to be a facet of P_C. Suppose this condition is not satisfied and let $k \in C$ be such that $k \in \ell(i) \cap \ell(j)$ for some $i, j \in H(C)$. Clearly, the constraint

$$x(H(C)) \leq |H(C)| - 2 + x_k \tag{12}$$

is valid for P_C since $x_k = 0$ implies $x_i = x_j = 0$ and therefore $x(H(C)) \leq |H(C)| - 2$ while for $x_k = 1$ (12) reduces to (6).

Rooting a valid inequality is the process of replacing a constant right-hand-side by a smaller constant and a positive weighted sum of additional variables. One form of valid inequality formed by rooting a minimal cover inequality (6) is related to the following structure.

Definition: Let $Z \subseteq V$ be such that $|Z|$ is odd and $\ell(Z)$ is a minimal cover. Z is a *zigzag* if it can be indexed from 1 to $|Z|$ so that for odd $i \in Z$, $\{i - 1, i + 1\} \subseteq \ell(i)$.

Given a zigzag Z we let Z^{odd} denote the set of elements of Z with odd indices and let Z^{even} denote the set of elements of Z with even indices. Further, we let $Z_i^+ = \{j \in Z : j \geq i\}$ and $Z_i^- = \{j \in Z : j \leq i\}$. It is notationally convenient, as in the definition of a zigzag, to allow implicit references to indices 0 and $|Z| + 1$ recognizing that conditions related to these elements are superfluous.

The following theorem is easily verified and is left to the reader.

Theorem 5 *If Z is a zigzag then the constraint*

$$x(Z^{odd}) \leq x(Z^{even}) \tag{13}$$

is valid for $P_{\ell(Z)}$.

Although zigzags do not always define facets, the zigzags captured by the following definition are facet defining.

Definition: A zigzag Z is *simple* if

$$|u(i) \cap Z^{odd}| = 1 \quad \text{for } i \in u(Z) \backslash Z^{even}$$
$$|u(i) \cap Z^{odd}| = 2 \quad \text{for } i \in Z^{even}$$

An example of a simple zigzag is depicted in Figure 1.

Theorem 6 *If a zigzag Z is simple then (13) is a facet of $P_{\ell(Z)}$.*

Proof: In order to complete the proof, let

$$\beta(\ell(Z)) \leq \omega \tag{14}$$

be such that $\beta(T) = \omega$ for all sets T with $x^T \in P_{\ell(Z)}$ and satisfying (13) at equality. We will show that (14) is a scalar multiple of (13), thus establishing (13) is a facet of $P_{\ell(Z)}$.

Suppose $i \in \ell(Z)$ is such that $\ell(i) \cap Z = \phi$. Clearly, both $x^{\ell(i)}$ and $x^{\ell(i)\backslash\{i\}}$ satisfy (14) at equality, implying $\beta_i = 0$.

Thus, suppose $i, j \in Z^{odd}$. Since $\ell(Z)$ is a minimal cover, $x^{\ell(Z)\backslash\{i\}}$ and $x^{\ell(Z)\backslash\{j\}}$ are both in $P_{\ell(Z)}$ and satisfy (14) at equality. As the sets $\ell(Z)\backslash\{i\}$ and $\ell(Z)\backslash\{j\}$ differ by a single element, it follows that $\beta_i = \beta_j$ for any two $i, j \in Z^{odd}$.

Next, consider $i \in u(Z)\backslash Z$. Since Z is simple, there exists a unique $j \in Z^{odd}$ such that $i \in \ell(j)$. Let $T = \ell(Z^-_{j-1}) \cup \ell(Z^+_{j+1}) \cup \ell(i)$. Since Z is simple both x^T and $x^{T\backslash\{i\}}$ are in $P_{\ell(Z)}$ and satisfy (14) at equality. As the sets T and $T\backslash\{i\}$ differ by a single element it follows that $\beta_i = 0$ for any $i \in u(Z)\backslash Z$.

Finally, consider $i \in Z^{even}$. Clearly, $x^{\ell(Z^-_i)}$ and $x^{\ell(Z^-_{i-2})}$ are both contained in $P_{\ell(Z)}$ and since Z is simple both satisfy (14) at equality. Of all $j \in \ell(Z^-_i)\backslash\ell(Z^-_{i-2})$ only $i-1$ and i have coefficients in (14) that have not been shown to be identically 0. It follows that $\beta_i = -\beta_{i-1}$. As all coefficients β_j for $j \in Z^{odd}$ have been shown to be identical, (14) must be a scalar multiple of (13), completing the proof. □

7 Single Root Facets

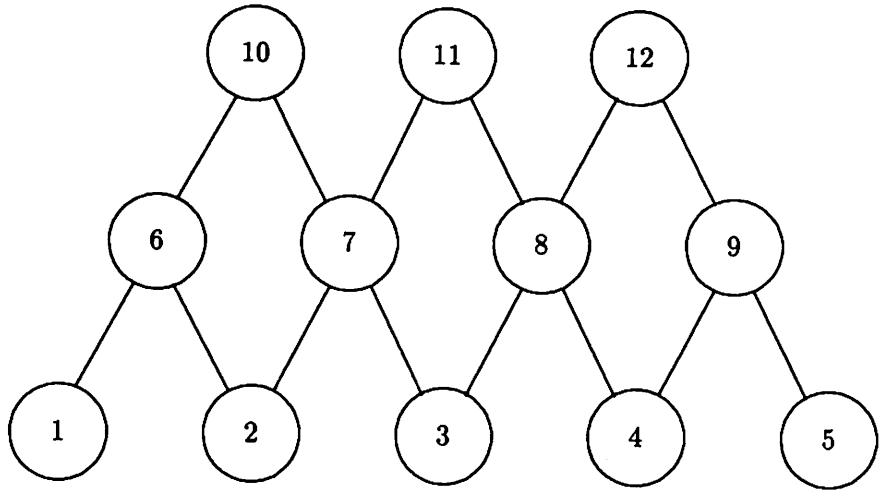
In the previous section zigzags were introduced to show how facets can be produced from constraints (6) that are not facets. The following theorem provides an alternative method of rooting constraints (6) to produce facets, but is not limited to minimal covers.

Theorem 7 *Let $S \in \mathcal{L}$ with $k \in S$ and assume $\sum_{i \in S\backslash\ell(k)} c_i x_i \leq \gamma$ with $\gamma > 0$ is a facet of $P_{S\backslash\ell(k)}$. If $k \in \ell(i)$ for all $i \in S\backslash\ell(k)$ with $c_i > 0$, then the constraint*

$$\sum_{i \in S\backslash\ell(k)} c_i x_i \leq \gamma x_k \tag{15}$$

is a facet of P_S with root k.

Proof: Since the constraint $\sum_{i \in S\backslash\ell(k)} c_i x_i \leq \gamma$ is valid for $P_{S\backslash\ell(k)}$ it is valid for P_S. By the assumption that $k \in \ell(i)$ for all $i \in S\backslash\ell(k)$ with $c_i > 0$, any $x^T \in P$ satisfying $\sum_{i \in S\backslash\ell(k)} c_i x^T_i > 0$ must have $x^T_k = 1$, implying (9) is valid for P_S. To prove that any such constraint is a facet of P_S, let S be indexed so as to satisfy Property 1 with the additional condition that if $i \in \ell(k)$ and $j \in S\backslash\ell(k)$ then $i < j$. Note that vertex k has index $|\ell(k)|$. Further, note that $P_{S\backslash\ell(k)}$ has dimension $|S\backslash\ell(k)|$ since the 0 vector together with the vectors $x^{\ell(i)\cap S}$, $i = |\ell(k)|+1, \ldots, |S|$, are affinely independent and are contained in $P_{S\backslash\ell(k)}$.



$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

Knapsack Constraint: $\sum_{i=1}^{12} x_i \leq 11$

$Z^{odd} = \{10, 11, 12\}$

$Z^{even} = \{7, 8\}$

Facet of $P_{t(Z)}$: $x_{10} + x_{11} + x_{12} \leq x_7 + x_8$

Figure 1

Example of a Zigzag

Construct an $|S| \times |S| - 1$ matrix M as follows. Let columns $i = 1, \ldots, |\ell(k)| - 1$ be the vectors $x^{\{1,\ldots,i\}}$. Let the remaining columns i be $x^{\ell(k) \cup Q(i)}$, where $Q(i) \in P_{S \setminus \ell(k)}$, $c(Q(i)) = \gamma$, and the vectors $x^{Q(i)}$ are affinely independent. The existence of $|S \setminus \ell(k)|$ such vectors follows from the assumption that $\sum_{i \in S \setminus \ell(k)} c_i x_i \leq \gamma$ is a facet of $P_{S \setminus \ell(k)}$. Since any set $Q(i)$ satisfying $c(Q(i)) = \gamma > 0$ must contain some j such that $c_j > 0$ and since $k \in \ell(j)$ by assumption, it follows that any $T \in \mathcal{L}$ with $a(T) \leq \alpha$ satisfying $T \cap (S \setminus \ell(k)) = Q(i)$ also satisfies $T \cap S = \ell(k) \cup Q(i)$, implying $x^{\ell(k) \cup Q(i)} \in P_S$. All of the columns in M therefore are contained in P_S and satisfy (15) at equality.

The $|\ell(k)| - 1 \times |\ell(k)| - 1$ upper-left submatrix of M is upper triangular with nonzero entries on the diagonal while the lower-left $|S| - (|\ell(k)| - 1) \times (|\ell(k)| - 1)$ matrix is the 0 matrix. It follows that any vector y satisfying $My = 0$ has $y_i = 0$ for $i = 1, \ldots, |\ell(k)| - 1$. The upper-right $|\ell(k)| \times |S| - |\ell(k)|$ submatrix is the 1 matrix and the lower-right $|S| - |\ell(k)| \times |S| - |\ell(k)|$ matrix consists of the $|S| - |\ell(k)|$ affinely independent vectors $x^{Q(i)}$. It follows that the remaining $|S| - |\ell(k)|$ columns are linearly independent and thus that the columns of M are linearly independent. Thus, the columns of M together with the 0 vector establish that (15) is a facet of P_S. □

An example of a rooted minimal cover is depicted in Figure 2.

8 Coalesced Minimal Covers

Let $C_p \in \mathcal{L}$ for $p = 1, \ldots, N$ be a collection of nonintersecting sets and let $C = \bigcup_{p=1}^{N} C_p$ be a minimal cover. Let $\mathcal{L}_{C_p}^- = \{S \subseteq \mathcal{L}_{C_p} : |S| \leq |C_p| - 1\}$ and note that $\mathcal{L}_{C_p}^-$ can be interpreted as the set of lower ideals in C_p that are feasible additions to the lower ideal $C \setminus C_p$. In addition to the conditions described above, we assume each of the sets C_p meets one of the following three conditions. The indexing of the sets C_p is to facilitate notation.

Condition 1:

C_p $p = 1$: $\ell(i) \cap \ell(j) = \phi$ for all $i, j \in H(C_1)$.

In this case

$$x(H(C_1)) \leq |H(C_1)| - 1 \tag{16}$$

is a facet of $P_{\mathcal{L}_{C_p}^-}$ (or contains $P_{\mathcal{L}_{C_p}^-}$ if $|H(C_1)| = 1$).

Condition 2:

C_p $p = 2, \ldots, n$: There exists a root $k(p)$ such that

$$x(H(C_p)) \leq (|H(C_p)| - 1) x_{k(p)} \tag{17}$$

is a facet of $P_{\mathcal{L}_{C_p}^-}$.

Condition 3:

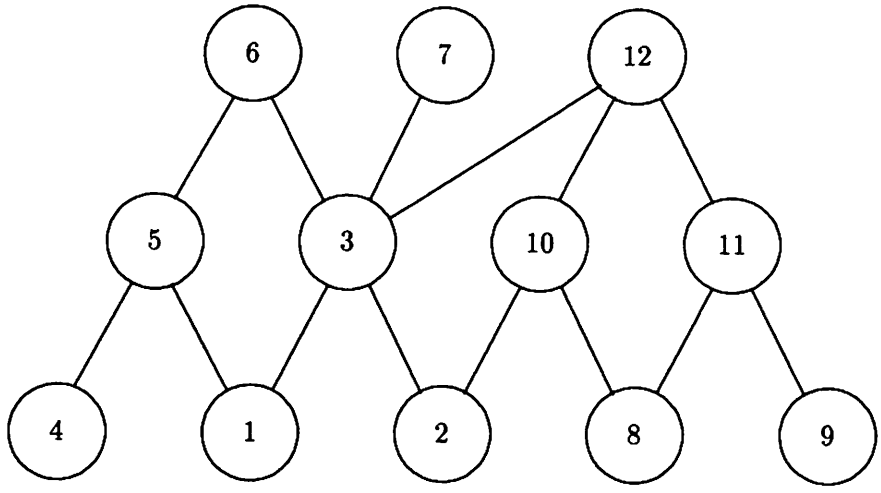
C_p $p = n+1, \ldots, N$: There exists a set Z_p with $|Z_p| = |H(C_p)| - 1$ such that $H(C_p) \cup Z_p$ is a simple zigzag and therefore

$$x(H(C_p)) \leq x(Z_p) \tag{18}$$

is a facet of $P_{\mathcal{L}_{C_p}^-}$.

In essence, these three conditions require that each set C_p demonstrate sufficient structure so that a facet for $P_{\mathcal{L}_{C_p}^-}$ can be found using the results in previous sections. Given the conditions just outlined, we define the following constraint.

$$\begin{aligned} x(H(C)) \leq (N-1) \quad & + |H(C_1)| - 1 \\ & + \textstyle\sum_{p=2}^{n} (|H(C_p)| - 1) x_{k(p)} \\ & + \textstyle\sum_{p=n+1}^{N} x(Z_p) \end{aligned} \tag{19}$$



$V = S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

Knapsack Constraint: $\sum_{i=1}^{12} x_i \leq 11$

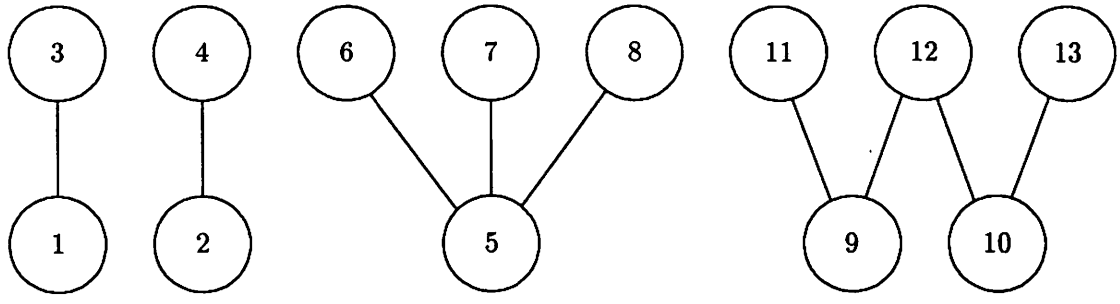
$k = 3$

Facet of $P_{S \setminus \ell(k)}$: $x_6 + x_7 + x_{12} \leq 2$

Facet of P_S: $x_6 + x_7 + x_{12} \leq 2x_3$

Figure 2

Example of a Rooted Minimal Cover



$V = C = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$

Knapsack Constraint: $\sum_{i=1}^{13} x_i \leq 12$

$N = 3$ $n = 2$

$C_1 = \{1, 2, 3, 4\}$

$C_2 = \{5, 6, 7, 8\}$

$C_3 = \{9, 10, 11, 12, 13\}$

Facet of P_C :

$$x_3 + x_4 + x_6 + x_7 + x_8 + x_{11} + x_{12} + x_{13} \leq 2 + 1 + 2x_5 + x_9 + x_{10}$$

Figure 3

Example of a Coalesced Minimal Cover

An example of a minimal cover satisfying the outlined conditions is depicted in Figure 3. Note that (19) is nothing more than the sum of the constraints (16), (17) and (18) with $N-1$ added to the right-hand-side.

Theorem 8 *Let $C_p \in \mathcal{L}$ for $p = 1, \ldots, N$ be a collection of nonintersecting sets, let $C = \bigcup_{p=1}^{N} C_p$ be a minimal cover, and let the C_p satisfy conditions (1)-(3). Then (19) is a facet of P_C.*

Proof: Let $T \in \mathcal{L}_C$ be such that $x^T \in P_C$. It is easily verified that each of the constraints (16), (17), and (18) can have a left-hand-side that exceeds the right-hand-side by at most 1 and that C_p is the only subset of $\mathcal{L}_{C_p}$ that achieves this violation. Further, because C is a minimal cover, and since C is the only subset of $\mathcal{L}_C$ violating all N constraints simultaneously, it follows that at most $N-1$ constraints can be violated by T. Since (19) is simply the sum of the constraints (16), (17), and (18) with $N-1$ added to the right-hand-side, it follows that (19) is valid for P_C.

To see that (19) is a facet of P_C, recall that except for the case $|H(C_1)| = 1$ the constraints (16), (17), and (18) are facets of their respective polyhedra $P_{\mathcal{L}_{C_p}}$. Since the $P_{\mathcal{L}_{C_p}}$ are of full dimension $|C_p|$, it follows that for each p there exist $|C_p|$ affinely independent vectors $x^{Q_p(i)}$ with $Q_p(i) \subseteq P_{\mathcal{L}_{C_p}}$ satisfying (16), (17), or (18) — depending on the value of p — at equality. For the special case $|H(C_1)| = 1$, $Q_1(i) = \ell(i)$ or 0 for all $i \in C_1 \backslash H(C_1)$ suffices. Further, it is easily verified that $x^{Q_p(i) \cup C \backslash C_p}$ satisfies (19) at equality.

Construct a matrix M as follows. Let the vertices in C be indexed so that they satisfy $i < j$ if $i \in C_p$, $j \in C_q$, and $p < q$. Let columns $1 + \sum_{q=1}^{p-1} |C_q|$ to $\sum_{q=1}^{p} |C_q|$ for $p = 1, \ldots, N$ correspond to the vectors $x^{Q_p(i) \cup C \backslash C_p}$. Note that the entries in M corresponding to the vectors $x^{Q_p(i)}$ form a collection of N block diagonal submatrices M_p of M with the rest of M comprised of 1's. Further, note that because the vectors $x^{Q_p(i)}$ for fixed p are affinely independent, the vectors $x^{Q_p(i) \cup C \backslash C_p}$ for a fixed p are linearly independent.

Let M' be the matrix formed from M as follows. For each $p = 1, \ldots, N$, choose any two rows of M that contain rows of M_p and subtract one row from the other. The resultant matrix M' has N rows that are identically 0 except where each intersects the columns of one of the matrices M_p. These rows together with the aforementioned linear independence of the vectors $x^{Q_p(i) \cup C \backslash C_p}$ for a fixed p imply that the only vector y satisfying $M'y = 0$ is the 0 vector. Thus, the columns of M are linearly independent, implying (19) is a facet of P_c. □

References

1. E. Balas, "Facets of the knapsack polytope," *Mathematical Programming* **8** (1975) 146-164.
2. E. Balas, "Facets of the knapsack polytope from minimal covers," *SIAM Journal of Applied Mathematics* **34:1** (1978) 119-148.
3. D.C. Cho, E.L. Johnson, M.W. Padberg, and M.R. Rao, "On the uncapacitated plant location problem I: Valid inequalities," *Mathematics of Operations Research* **8** (1983) 579-589.
4. D.C. Cho, E.L. Johnson, M.W. Padberg, and M.R. Rao, "On the uncapacitated plant location problem II: Facets and lifting theorems," *Mathematics of Operations Research* **8** (1983) 590-612.
5. G. Cornuejols and J.M. Thizy, "Some facets of the simple plant location polytope," *Mathematical Programming* **23** (1982) 50-74.
6. H. Crowder, E.L. Johnson, and M.W. Padberg, "Solving large-scale zero-one linear programming problems," *Operations Research* **31** (1983) 803-834.

7. P.L. Hammer, E.L. Johnson, and U.N. Peled,"Facets of regular 0-1 polytopes," *Mathematical Programming* **8** (1975) 179-206.

8. P.L. Hammer and B. Simeone, "Order relations of variables in 0-1 programming," *Annals of Discrete Mathematics* **31** (1987) 83-112.

9. D. Hartvigsen and E. Zemel, "On the complexity of lifted inequalities for the knapsack problem," Discussion Paper 740, Department of Managerial Economics and Decision Sciences, Northwestern University (Evanston, IL, 1987).

10. B. Korte and L. Lovász, "Polyhedral results for antimatroids," Report 85390-OR, Institute of Operations Research, University of Bonn (Bonn, West Germany, 1985).

11. J. Krarup and P.M. Pruzan, "The simple plant location problem: Survey and synthesis," *European Journal of Operational Research* **12** (1983) 36-81.

12. J.M.Y. Leung and T.L. Magnanti, "Valid inequalities and facets of the capacitated plant location problem," *Mathematical Programming* **44:3** (1989) 271-292.

13. G. Nemhauser and L.E. Trotter, "Properties of vertex packing and independence system polyhedra," *Mathematical Programming* **6** (1974) 48-61.

14. G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization* (John Wiley and Sons, New York, NY, 1988).

15. M.W. Padberg, "On the facial structure of set packing polyhedra," *Mathematical Programming* **5** (1973) 199-215.

16. M.W. Padberg, "A note on zero-one programming," *Operations Research* **23** (1975) 833-837.

17. M.W. Padberg, "Covering, packing, and knapsack polytopes," *Annals of Discrete Mathematics* **4** (1979) 265-287.

18. J.C. Picard and M. Queyranne, "Selected applications of minimum cuts in networks," *INFOR* **20:4** (1982) 394-422.

19. R. Sridharan "A survey of the capacitated plant location problem," GSIA, Carnegie-Mellon University (Pittsburgh, PA, 1984).

20. L. Wolsey, "Faces for a linear inequality in 0-1 variables," *Mathematical Programming* **8** (1975) 165-178.

21. E. Zemel, "Easily computable facets of the knapsack polytope," Working Paper 713, Department of Managerial Economics and Decision Sciences, Northwestern University (Evanston, IL, 1987).

MODULAR ARITHMETIC AND RANDOMIZATION FOR EXACT MATROID PROBLEMS*

P.M. Camerini
Università di Siena, Italy

G. Galbiati
Università di Pavia, Italy

F. Maffioli
Politecnico di Milano, Italy

Abstract

Random pseudo-polynomial algorithms working in modular arithmetic are presented for the problem of finding a base of given value in a represented matroid subject to parity conditions with set of elements E weighted in the integers. Assuming Chowla's conjecture, testing for the existence of such a base requires $O(n^2\ m + n^3\ U\ (n + \log\ U))$ arithmetic operations, where n is the rank of the matroid, $m = \mid E \mid$ and U is the maximum integral weight of an element. Some particular cases are also briefly discussed.

1 Introduction

In the last years attention has been devoted to the study of *exact* combinatorial problems, i.e. problems which, given a weighted set E of elements and a family of feasible subsets of E, look for a feasible subset such that the sum of the weights of its elements equals a given target value [BaP87, MVV87, CGM87, CGM89]. Some exact combinatorial problems can be solved using *random pseudo-polynomial* (r.p.p.) algorithms, i.e. algorithms which work in pseudo-polynomial time and which always answer correctly in the case of a no-instance; for a yes-instance they may yield a wrong answer, with probability not greater than any preassigned constant $\varepsilon = \frac{1}{\rho}$, $\varepsilon < 1$, independent of the input size. Specifically, in [CGM89] r.p.p. algorithms are presented for solving the **Exact Parity Base** (XPB) problem, a problem which asks for a base of given value in a weighted represented matroid subject to parity conditions. In the same work two special cases of XPB having specific interest have also been addressed: the problem of finding a base of exact value in the intersection of two (represented) matroids and the problem of finding a perfect matching of exact value in a weighted graph. These two problems are called **Exact Intersection Base** (XIB) and **Exact Perfect Matching** (XPM), respectively.

*Partially sponsored by research contracts MPI 40% and 60% of the Italian Ministry of University and Scientific Research and by NATO Grant RG 85-0240.

In this paper we present faster algorithms for solving the same problems. Similarly to the algorithms presented in [CGM89], these new algorithms are sequential and random pseudo-polynomial. The improvements we have obtained, with respect to the algorithms presented in [CGM89], are due to better interpolation techniques, more extensive randomization, and faster arithmetics working on suitably small finite fields. In Section 2 we present some algebraic and numerical results; in Section 3 we describe and analyze four different r.p.p. algorithms for solving the XPB problem; these algorithms are compared in Section 4, where we also briefly discuss the solution algorithms for the XPM and XIB problems.

2 Algebraic and numerical results

Let us begin by recalling a result due to Schwartz, which yields a probabilistic method for testing whether a polynomial has modular zeros.

Theorem 1 [Sch80] Let $Q(x)$, $x = (x_1, \ldots, x_m)$, be a polynomial of degree n in m variables. Let $\{q_1, \ldots, q_H\}$ be a set of primes such that the product of any $h-1$ of them exceeds the maximum value that $Q(x)$ can achieve in $\{1, \ldots, J\}^m$, $J \in N$. Let q be chosen randomly in $\{q_1, \ldots, q_H\}$ and $\bar{x}$ be chosen randomly in $\{1, \ldots, J\}^m$. If $Q(x) \not\equiv 0$, then

$$\text{Prob}\,\{Q(\bar{x} \equiv 0 \bmod q\} \leq \frac{n}{J} + \frac{h}{H}.$$

The second result we need is derived from the following well-known theorem.

Prime Number Theorem [Cha68] Let q_n be the n-th prime number. Then

$$\lim_{n\to\infty} \frac{q_n}{n \log_e n} = 1.$$

From this theorem it follows that there exists a constant k such that, for $n > 1$, $q_n \leq k\, n \log_e n$. In Chapter VII of [Cha68] it is proved that k can be chosen equal to $\frac{3}{2 \log_e 2}$. We may therefore obtain the following upper bound for q_n:

$$q_n \leq \frac{3}{2}\, n \log_2 n. \tag{1}$$

The next result we need is an estimate of the number of arithmetic operations required for generating all primes in a given interval $[2, \alpha]$, α integer.

Proposition 1 The generation of all primes in $[2, \alpha]$, α integer, by Eratostene's sieve requires $O(\alpha \log \alpha)$ arithmetic operations on operands of magnitude smaller than 2α.
This Proposition is easily proved by noticing that the number of necessary operations is $O(\alpha + \frac{\alpha}{2} + \frac{\alpha}{3} + \ldots + \frac{\alpha}{\alpha}) = O(\alpha \log \alpha)$.

The next result has been communicated to the authors by a specialist in the field of Analytic Number Theory [Vio88].

Theorem 2 Let q_n be the n-th prime congruent to 1 mod p, with p prime. Then

$$q_n = O(p^L \ n \ \log \ n)$$

where $L = \alpha + \varepsilon_1, \varepsilon_1 > 0$, is the Linnik's constant [Lin44].
For what concerns the value of L, Gallagher [Gal72] has proved that, if p is prime, then $L = \frac{5}{2} + \varepsilon_1$, in [BLT64] it is proved that, on the assumption of the "extended Riemann hypothesis", $L = 2 + \varepsilon_1$; Chowla [Cho34] has conjectured that $L = 1 + \varepsilon_1$.

We now quote two final results which are concerned with finding primitive roots and inverses in finite fields.

Theorem 3 If p is prime, a primitive root of order p in the field Z_{kp+1}, k integer, can be found in $O(k^2 \ \log \ k + \log \ p)$ arithmetic operations in the field.

Proof The proof consists in producing an algorithm that computes the primitive root.

This algorithm is similar to Gauss'algorithm [McE87] for finding a primitive root in any field, but improves over it in terms of number of operations.
The algorithm is given as procedure PROOT(p,k) below, and uses procedure PROCESS (α,φ,t), which given α in Z_{kp+1} returns $\varphi = 0$ and the order of α if $t \leq k$, otherwise it returns $\varphi = 1$ and an integer t such that α^t is a primitive root of order p.

procedure PROOT (p,k) :
1. find a non-zero element α in Z_{kp+1};
2. PROCESS(α,φ,t);
3. **if** $\varphi = 1$ **then return** $\omega = \alpha^t$;
   comment $t \leq k$ is the order of α ;
   repeat
4. find a non-zero element β in Z_{kp+1}, which is not a power of α;
5. PROCESS(β,φ,s);
6. **if** $\varphi = 1$ **then return** $\omega = \beta^s$;
   comment $s \leq k$ is the order of β and the steps to follow
   compute as in Gauss' algorithm a new element whose order is a multiple of t;
7. find a divisor d of t and a divisor e of s such that $gcd(d,e) = 1$ and
   $\text{lcm}(t,s) = de$;
8. $\alpha := \alpha^{t/d} \ \beta^{s/e}$;
9. $t := d \ e$
   until $t > k$;
   comment the order of α is a multiple of p;
10. $j := t/p$;
11. **return** $\omega = \alpha^j$

procedure PROCESS(α, φ, t) :
compute at most k successive powers of α until either $\alpha^t = 1$ for some t
or $\alpha^t \neq 1, t = 1, \ldots, k$;
if for $t = 1, \ldots, k$, $\alpha^t \neq 1$ **then**
begin
    comment the order of α is a multiple of p;
    compute $\gamma = \alpha^p$;
    compute successive powers of γ until, for some t ($1 \leq t \leq k$),
    $\gamma^t = 1$;
    $\varphi = 1$;
    return φ, t
end ;
$\varphi := 0$;
return φ, t

It is easy to see that PROCESS requires $O(k)$ operations in the field if it returns $\varphi = 0$, otherwise it requires $O(k + \log\ p)$ operations. In procedure PROOT the number of iterations of the **repeat** loop is $O(\log\ k)$, since in each iteration α is updated and its order at least doubles. Since line 4 requires $O(k^2)$ operations the overall number of arithmetic operations required by the algorithm is $O(k^2\ \log\ k + \log\ p)$. □

Proposition 2 The inverse of a non-zero element α in a field Z_q can be found in $O(\log\ q)$ operations in the field.

Sketched proof The proof follows from the fact that constants a and b can be found, using Euclid's algorithm, such that $a\ \alpha + b\ q = \gcd(\alpha, q)$. Since q is prime, obviously $\gcd(\alpha, q) = 1$ and the inverse of α is a. □

3 Algorithms for the XPB Problem

In this Section we present and analyze four algorithms for solving the XPB problem, a problem thouroughly discussed in [CGM89] where the reader is invited to refer for a more complete presentation of it. The XPB problem can be stated as follows.

EXACT PARITY BASE (XPB)

Instance: - a matroid represented by the integer matrix
$M = (a_1 \mid b_1 \mid \ldots \mid a_m \mid b_m)$, of rank $2n$,
whose columns are partitioned into m pairs $\{a_j, b_j\}$,
$j \in E = \{1, \ldots, m\}$;
- a non-negative integer weight w_j, assigned to the j-th pair, $j \in E$;
- a non-negative integer W.

Question: - does there exist a parity base of M whose weight is exactly W,
i.e. a set $X = \{j_1, \ldots, j_n\} \subseteq E$ such that
$\{a_{j_1} \mid b_{j_1} \mid \ldots \mid a_{j_n} \mid b_{j_n}\}$ has rank $2n$ and
$\sum_{j \in X} w_j = W$?

The algorithms described in [CGM89] for solving XPB are based on a methodology that:

A. associates to each instance of the problem a $2n$ x $2n$ skew-symmetric matrix C, defined as follows:

$$C = \sum_{j=1}^{m} x_j \, y^{w_j} \, (a_j \wedge b_j), \tag{2}$$

where $a_j \wedge b_j$ is the *wedge product* of the two column vectors a_j, b_j, defined as the $2n$ x $2n$ skew-symmetric matrix $a_j \, b_j^T - b_j \, a_j^T$;

B. proves that the following properties holds for the pfaffian of matrix C:

1. $\mathrm{pf}(C)$ is a polynomial in the variables $x_1, \ldots, x_m, y$, which can be written as

$$\sum_{1 \le j_1 < \ldots < j_n \le m} x_{j_1} \ldots x_{j_n} \, y^{\omega(j_1, \ldots, j_n)} \, \delta(j_1, \ldots, j_n) \tag{3}$$

with $\omega(j_1, \ldots, j_n) = \sum_{1 \le h \le n} w_{j_h}$ and

$$\delta(j_1, \ldots, j_n) = \det(a_{j_1} \mid b_{j_1} \mid \ldots \mid a_{j_n} \mid b_{j_n}) \tag{4}$$

2. each monomial of $\mathrm{pf}(C)$ corresponds to a parity base of M;
3. the degree of y in each monomial is the weight of the corresponding parity base;

C. expresses the pfaffian of C as the following polynomial in y:

$$P(x, y) = \mathrm{pf}(C(x, y)) = \sum_{t=0}^{p-1} Q_t(x) \, y^t, \tag{5}$$

where $x = (x_1, \ldots, x_m)$, $p - 1$ is any upper bound to the maximum weight of a parity base, and each coefficient $Q_t(x)$ is a polynomial in $x_1, \ldots, x_m$ sum of all terms

$$x_{j_1} \ldots x_{j_n} \, \delta(j_1, \ldots, j_n)$$

with $1 \le j_1 < \ldots < j_n \le m$ corresponding to parity bases of weight t;

D. concludes that there exists a parity base of weight W if and only if $Q_W(x) \not\equiv 0$;

E. tests whether $Q_W(x) \not\equiv 0$ by computing $Q_W(\bar{x})$ for some vector $\bar{x}$ chosen randomly in $\{1, \ldots, J\}^m$.

The algorithms we are going to present here follow a similar methodology, the substantial difference being that each of them:

(i) works on a finite field chosen at random among a set of suitably constructed finite fields,

(ii) computes in the field the coefficient $Q_W(\bar{x})$ by a method appropriate to the field.

The backbone of the four algorithms is given by procedure XPBM below, where lines 5, 6, and 9 are left partially undefined, since it is by completely specifying these lines that the four algorithms are obtained. It is important to specify here that matrix $C(x,y)$ in line 8 is defined exactly as in (2), that $w = (w_1,\ldots,w_m)$, and ρ is an assigned constant.

procedure XPBM(n,m,M,w,W,ρ):

1. find an integer p larger than the maximum weight of a parity base of M ;
2. **if** $W \geq p$ **then return** 'NO' ;
3. $J := 2n\rho$;
4. choose randomly $\bar{x}$ in $\{1,\ldots,J\}^m$;
5. compute an appropriate integer value H ;
6. generate an appropriate set $\mathcal{H} = \{q_1,\ldots,q_H\}$ of primes;
7. choose randomly q in $\mathcal{H}$;
8. compute in Z_q the $2n$ x $2n$ skew-symmetric matrix $C(\bar{x},y)$;
9. compute in Z_q the coefficient $Q_W(\bar{x})$ of the term y^W of $P(\bar{x},y) = \mathrm{pf}(C(\bar{x},y))$;
10. **return if** $Q_W(\bar{x}) = 0$ in Z_q **then** 'NO' **else** 'YES'.

The reader is invited to analyze and compare procedure XPB of [CGM89] with this procedure, and to note that lines 5, 6 and 7 constitute the relevant new feature, to which we are going to devote our attention.

We begin by proving that, for suitable choices of H and $\mathcal{H}$ in lines 5 and 6, procedure XPBM satisfies the requirements on the error performance of a r.p.p. algorithm. Since the procedure is based on the methodology expressed in A, B, C, D, E it is easy to conclude that if the matroid has no parity base of weight W, then $Q_W(x) \equiv 0$ in Z_q and procedure XPBM always answers correctly. On the other hand if such a parity base exists the answer of the procedure may be wrong but the probability of a wrong answer in this latter case is that of the event $Q_W(\bar{x}) = 0$ in Z_q, given $Q_W(x) \not\equiv 0$. By theorem 1, this probability is bounded by $\frac{n}{J} + \frac{h}{H}$.

We now show that $\frac{n}{J} + \frac{h}{H} \leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon$, $\varepsilon = \frac{1}{\rho}$, and that the hypothesis of Theorem 1 are satisfied hence completing the proof of the correctness of XPBM. It is straightforward to see that the choice of J in line 3 implies $\frac{n}{J} \leq \frac{\varepsilon}{2}$; we will see in the next paragraphs that, in line 5, H will always be chosen equal to $\lceil \frac{2h}{\varepsilon} \rceil$, for some specific value of h, thus implying that $\frac{h}{H} \leq \frac{\varepsilon}{2}$. We now turn our attention to the hypothesis of Theorem 1. In the first three algorithms which generate a set $\mathcal{H}$ of distinct primes not smaller than p, we will set $h = \log_p(2JV^2\ n\ m)^n + 1$, V being the maximum absolute value of an element of M, whereas in the fourth algorithm, which generates a set $\mathcal{H}$ of distinct primes not smaller than 2, we will set $h = \log_2(2JV^2\ n\ m)^n + 1$. Obviously in the first three algorithms the product of any $h-1$ primes is not smaller than α, with $\alpha = p^{h-1}$, whereas in the forth algorithm such product is not smaller than α, with $\alpha = 2^{h-1}$. From the different settings of h, it follows that in all cases α is not smaller than $(2JV^2\ n\ m)^n$; this last value turns out to be

an upper bound to the maximum value that polynomial $Q_W(x)$ can achieve in $\{1,\ldots,J\}^m$ since from (3) and (5) we can deduce that the following inequalities hold if $x \in \{1,\ldots,J\}^m$:

$$Q_W(x) \leq \binom{m}{n} J^n(V^{2n}(2n)!) = J^n V^{2n}\frac{(2n)!}{n!}\frac{m!}{(m-n)!} < J^n V^{2n}(2n)^n m^n.$$

This is enough to conclude that the hypothesis of Theorem 1 are satisfied and that procedure XPBM is correct.

In the remaining paragraphs of this Section the time complexities of the four algorithms will be analyzed and shown to be always pseudo-polynomial. The time will be measured both in arithmetic and in bit operations. The arithmetic operations in the first seven lines of XPBM are operations in the ring of integers, whereas those in the remaining lines are operations in the chosen finite field. In all paragraphs we assume for the sake of simplicity that each arithmetic operation on operands of magnitude μ requires $O(\log\ \mu)$ bit operations, and that $W = O(p)$, that is W is not unboundely larger than the maximum weight that a parity base can achieve.

3.1 Algorithm 1

The first algorithm we present here is similar to the first solution algorithm proposed in [CGM89] for the XPB problem, but, differently from that, it works in a finite field Z_q having at least p elements. We describe the algorithm by specifying lines 5, 6 and 9 of procedure XPBM as follows.

5. let

$$H := \lceil \frac{2}{\varepsilon}(\log_p(2JV^2\ n\ m)^n + 1 \rceil; \tag{6}$$

6. generate the set $\mathcal{H} = \{q_1,\ldots,q_H\}$ of the first H primes greater or equal to p;

9. evaluate in Z_q, $P(\bar{x},y) = \mathrm{pf}(C(\bar{x},y))$ for $y = 0,\ldots,p-1$ and interpolate using Newton's classical methodology, computing in Z_q the coefficient $Q_W(\bar{x})$ of the term y^W of $P(\bar{x},y)$;

Before analyzing the time complexity of this procedure we estimate the number b of bits necessary for encoding each element of the chosen field Z_q. From (6) it is easy to deduce that $H = O(n\ \frac{\log\ n + \log\ V + \log\ m}{\log\ p})$; if at this point we make the simplifying yet realistic hypothesis, already made in [CGM89], that

$$(\log\ V + \log\ m) = O(\log\ p) \tag{7}$$

or the more restrictive hypothesis that:

$$(\log\ V + \log\ m) = O(\log\ n) \tag{8}$$

we may conclude that

$$H = O(n).$$

By the Prime Number Theorem we may deduce that the biggest prime q_H generated in line 6 satisfies the inequality:

$$q_H \leq 1{,}5(p+H)\log_2(p+H). \tag{9}$$

and therefore that the number b of bits necessary for encoding any element in a field Z_q where q is chosen at random in $\mathcal{H}$, satisfies the equality:

$$b = \lceil \log_2 q_H \rceil = O(\log p). \tag{10}$$

Note that the result in (10) is asymptotically optimum, since in order to evaluate and interpolate in a field a polynomial of degree $p-1$, the field must contain at least p elements.

We turn now our attention to the time complexity of the algorithm measured either in arithmetic or in bit operations. The next theorem shows that if we measure the time in arithmetic operations the complexity of algorithm 1 is the same as that of the first algorithm proposed in [CGM89], whereas if we use bit operations the complexity significantly improves, since the number of bits involved in each arithmetic operation decreases to the asymptotically optimum order $O(\log p)$.

Theorem 4 If (7) or (8) hold, the complexity of Algorithm 1 is

$$O_A(p\, n^2\, m \log U + p^2), \tag{11}$$

or

$$O_B((p\, n^2\, m \log U + p^2)\log p) \tag{12}$$

depending on whether the time is measured in arithmetic or bit operations, respectively.

Proof We begin by observing that, for the sake of simplicity, line 1 can be implemented by setting $p = n\, U + 1$, with $U = \max\{w_j, j \in E\}$, thus using $O(m)$ arithmetic operations on integers not greater than p. Line 2 requires $O(1)$ operations on operands of magnitude $O(p)$. Obviously lines 3, 4 and 5 require $O(m)$ arithmetic operations on operands of magnitude $O(n)$. Because of Proposition 1, line 6 can be implemented using $O(q_H \log q_H)$ arithmetic operations on operands of magnitude of order $O(q_H)$; from (10) this number of arithmetic operations simplifies to $O(p \log^2 p)$ and the magnitude of the operands simplifies to $O(p \log p)$. Obviously, line 7 requires $O(1)$ arithmetic operations on operands of magnitude $O(p \log p)$. The computations in line 8 require $O(n\, m)$ operations on operands of magnitude $O(V)$ to compute in the chosen field the elements of M,

plus $O(m)$ operations on operands of magnitude $O(J)$ to compute in the field the m components of $\bar{x}$, plus $O(n^2\ m)$ operations in the field for computing the m non-zero coefficients of each of the n^2 polynomials in matrix $C(\bar{x}, y)$. Finally line 9 can be implemented using $O(p\ n^2\ m\ \log\ U)$ operations in the field for computing the values that the n^2 polynomials (of degree at most U) achieve for each value $y = 0, \ldots, p-1$, plus $O(p\ n^3)$ operation in the field for the evaluation of the p pfaffians with the algorithm described in [CGM87], plus $O(p^2)$ operations in the field for the subsequent interpolation. The theorem follows now easily from (10). □

3.2 Algorithm 2

This second algorithm is identical to Algorithm 1 but for line 9 where, instead of Newton's classical interpolation method, it uses the more sophisticated interpolation method described in [AHU74] (p. 299) which requires only $O(p\ \log^2\ p)$ operations in the field instead of the $O(p^2)$ operations of Newton's method. It is therefore straightforward to prove the following theorem.

Theorem 5 If (7) or (8) hold, the complexity of Algorithm 2 is

$$O_A(p\ n^2\ m\ \log\ U + p\ \log^2\ p) \tag{13}$$

or

$$O_B((p\ n^2\ m\ \log\ U + p\ \log^2\ p) \log\ p) \tag{14}$$

depending on whether the time is measured in arithmetic or bit operations, respectively.

3.3 Algorithm 3

This algorithm has many features which make it substantially different from the two preceding ones. After having increased the value of p to make it become a prime, the algorithm generates a set of primes of type $k\ p + 1$, k integer greater or equal to 1. Since in a field Z_q, with $q = k\ p + 1$, there exists a primitive root of order p, the algorithm finds such root and uses the discrete direct and inverse Fast Fourier Transform (FFT) as evaluation and interpolation method in line 9. We describe the algorithm again by specifying lines 5, 6 and 9 of procedure XPBM.

5. $H := \lceil \frac{2}{\varepsilon}(\log_p(2JV^2 n\ m)^n + 1) \rceil$;
6. increase the value of p to become prime and generate the set $\mathcal{H} = \{q_1, \ldots, q_H\}$ of the first H primes of the kind $k\ p + 1$ with k integer $\geq\ 1$;
9. find in Z_q a primitive root ω of order p, evaluate in Z_q each entry of $C(\bar{x}, y)$ for $y = \omega^0, \omega^1, \ldots, \omega^{p-1}$ using the direct FFT, compute in Z_q $P(\bar{x},\ \omega^j) = \mathrm{pf}(C(\bar{x}, \omega^j)$ for $j = 0, \ldots, p-1$, and interpolate using the inverse FFT, computing in Z_q the coefficient $Q_W(\bar{x})$ of the term y^W of $P(\bar{x}, y)$ with the well-known formula

$$Q_W(\bar{x}) = p^{-1} \sum_{j=0}^{p-1} P(\bar{x}, \omega^j)\omega^{-jW} \tag{15}$$

We now analyze the time complexity of the procedure. Since line 5 is as in (6) we know that $H = O(n)$. Increasing p to become prime in line 6 by Eratostene's sieve accounts for $O(p \log p)$ arithmetic operations on operands of magnitude $O(p)$. The generation of the set $\mathcal{H} = \{q_1 = k_1 p + 1, \ldots, q_H = k_H p + 1\}$ of primes can be done by testing the primality of every integer of the kind $k p + 1$, for increasing even integers k. These tests require $O(k_H \sqrt{q_H})$ arithmetic operations on operands of magnitude $O(q_H)$. Because of Theorem 2 we know that

$$q_H = O(p^{\alpha+\epsilon_1} H \log H)$$

and therefore obtain for generating $\mathcal{H}$ an overall number of arithmetic operations

$$O(p^{\frac{3}{2}\alpha-1+\frac{3}{2}\epsilon_1} n^{\frac{3}{2}} \log^{\frac{3}{2}} n)$$

Using Theorem 3 and the modified Gauss' algorithm mentioned in its sketched proof, we can find a primitive root ω of order p in the chosen field with

$$O(p^{2\alpha-2+2\epsilon_1} n^2 \log^2 n \log p)$$

arithmetic operations in the field. The remaining part of line 9 requires $O(p\, n^2 \log p)$ operations in the field for computing the values of the entries of matrix $C(\bar{x}, y)$ for all $y = \omega^o, \omega^1, \ldots, \omega^{p-1}$, plus $O(p\, n^3)$ operations for computing the pfaffians, and $O(p)$ operations for applying (15) (because of Proposition 2). This analysis of the complexity of Algorithm 3 allows to state the following theorem.

Theorem 6 If Chowla's conjecture is true and (7) or (8) hold, the complexity of Algorithm 3 is

$$O_A(n^2 m + p\, n^2 \log p + p\, n^3) \tag{16}$$

or

$$O_B((n^2 m + p\, n^2 \log p + p\, n^3) \log p) \tag{17}$$

depending on whether the time is measured in arithmetic or bit operations respectively. □

3.4 Algorithm 4

This algorithm is similar to the second solution algorithm described in [CGM89] for the XPB problem, but differently from that, it works in a finite field Z_q to which we do not need to require to have at least p elements. Lines 5, 6 and 9 of XPBM are specified as follows

5. $H := \lceil \frac{2}{\epsilon} \log_2 (2JV^2\ n\ m)^n + 1 \rceil$;
6. generate the set $\mathcal{H} = \{q_1, \ldots, q_H\}$ of the first H primes;
9. evaluate in Z_q, $P(\bar{x}, y) = \mathrm{pf}(C(\bar{x}, y)$ directly applying the algorithm in [CGM87] to the matrix of polynomials with coefficient in the field chosen in line 8;

From line 5 we have that, if the hypothesis in (7) holds, then $H = O(n \log p)$, whereas if the hypothesis in (8) holds, then $H = O(n \log n)$. From (1) we know that

$$q_H \leq 1.5\ H \log_2 H$$

and therefore that the number b of bits necessary for encoding any element in a field Z_q, with q chosen at random in $\mathcal{H}$, satisfies the equality $b = O(\log n + \log \log p)$ or $b = O(\log n)$ depending on whether (7) or (8) holds. The complexity of Algorithm 4 results from the following theorem.

Theorem 7 The complexity of algorithm 4, measured in artihmetic operations, is

$$O_A(n^2\ m + p\ n^3 \log p). \tag{18}$$

The complexity, measured in bit operations, is

$$O_B(n\ m \log U + (n^2\ m + p\ n^3 \log p)(\log n + \log \log p)) \tag{19}$$

or

$$O_B(n\ m \log U + (n^2\ m + p\ n^3 \log p) \log n) \tag{20}$$

depending on whether the hypothesis in (7) or in (8) holds.

Proof Simple considerations allow to conclude that the computations in lines 8 and 9 determine the overall complexity of the algorithm. The computations in line 8 require exactly the same time as in algorithm 1. Line 9 can be implemented using $O(p\ n^3 \log p)$ operations in the field since $O(n^3)$ operations between polynomials of degree at most $p-1$ are required and each of these operations requires at most $O(p \log p)$ time [AHU74]. From these considerations (18) follow immediately, whereas (19) and (20) follow since, depending on whether (7) or (8) holds, $\log V$ is $O(\log p)$ or $O(\log n)$ and $\log H$ is $O(\log n + \log \log p)$ or $O(\log n)$. □

4 Algorithms comparison and final remarks

On the basis of theorems 4, 5, 6 and 7, we can make some considerations on the relative performances of the four algorithms proposed for the XPB problem. If the complexity is measured in arithmetic

operations it is easy to see that if Chowla's conjecture is true, algorithm 3 achieves the best performance. Assuming instead that such conjecture is false, then if $n \log n = O(m \log U)$ algorithm 4 is faster than both algorithm 2 and algorithm 1, whereas if $n \log n = \Omega(m \log U)$, it can be seen that Algorithm 2 performs better than both algorithms 1 and 4. Note however that algorithm 1 is by far the simplest to implement. A more cumbersome situation emerges when comparing the complexities measured in bit operations, and is not reported here for the sake of brevity. Constructive versions of all the algorithms may be designed as in [CGM89].

We turn now our attention to the two special cases of the XPB problem mentioned in the Introduction, namely problems XIB and XPM, but we devote to them only a brief discussion. For the sake of brevity we do not present the four algorithms for these problems.

As in [CGM89] the algorithms for XIB differ from the corresponding ones for XPB only because they work with a different matrix, but it can be proved that their overall complexities remain the same as expressed in theorems 4, 5, 6 and 7.

As in [CGM89] the algorithms for problem XPM work on the simpler matrix $C = (c_{ij})$, where

$$c_{ij} = \begin{cases} y^{w_e}\, x_e & \text{if } e = \{i,j\} \in E \text{ and } i < j \\ -y^{w_e}\, x_e & \text{if } e = \{i,j\} \in E \text{ and } i > j \\ 0 & \text{otherwise,} \end{cases}$$

$G = (\{1, 2, \ldots, 2n\}, E\}$ being the given weighted graph.

A substantial improvement in the complexities of the four algorithms for XPM can thus be achieved as expressed in the following theorem which we state without proof.

Theorem 8 The complexities of the algorithms 1, 2, 3 and 4 for XPM are

$$O_A(p\, n^2 \log U + p\, n^3 + p^2),$$

$$O_A(p\, n^2 \log U + p\, n^3 + p \log^2 p),$$

$$O_A(p\, n^2 \log U + p\, n^3),$$

$$O_A(p\, n^3 \log p)$$

respectively, when measured in arithmetic operations. Under hypothesis (7) or (8) all arithmetic operations in the first three algorithms are on operands requiring $O(\log p)$ bits, whereas in the fourth algorithm they require $O(\log n)$ bits.

References

AHU74 A.V. AHO, J.E. HOPCROFT, and J.D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading MA, 1974.

BaP87 F. BARAHONA and W.R. PULLEYBLANK, "Exact Arborescences, Matching and Cycles", *Discrete Appl. Math.* **16** (1987) 91-100.

BLT64 M.B. BARBAN, YU. V. LINNIK, and N.G. TSCHUDAKOV, "On the Prime Numbers in an Arithmetic Progression with a Prime-Power Difference", *Acta Arithmetica* **9** (1964) 375-390.

CGM87 P.M. CAMERINI, G. GALBIATI and F. MAFFIOLI, Random Pseudo- Polynomial Algorithms and Exact Problems, Int. Rep. n. 87-014, Dip. di Elettronica, Politecnico di Milano (1987).

CGM89 P.M. CAMERINI, G. GALBIATI and F. MAFFIOLI, "Random Pseudo- polynomial Algorithms for Exact Matroid Problems", (submitted to *J. of Algorithms*).

Cha68 K. CHANDRASEKHARAN, *Introduction to Analytic Number Theory*, Springer- Verlag 1968.

Cho34 S. CHOWLA, "On the Least Prime in an Arithmetic Progression", *J. Indian Math. Soc.* N.s. 1, 1-3 (1934).

Gal72 P.X. GALLAGHER, "Primes in Progressions to Prime-Power Modulus", *Inventiones Math.* **16** (1972) 191-201.

KUW85 R.M. KARP, E. UPFAL and A. WIDGERSON, "Finding a Maximum Matchingis in RNC", *Proc. 17th ACM Symp. on the Th. of Computing* (1985) 22-32.

Lin44 YU. V. LINNIK, "On the Least Prime in an Arithmetic Progression", *Mathem. Sbornik* **15** (57) (1944) 139-178 and 347-368 (in russian).

McE87 R.J. MCELIECE, *Finite Fields for Computer Scientists and Engineers*, Kluwer Ac. Publ., 1987.

MVV87 K. MULMULEY, U.V. VAZIRANI and V.V. VAZIRANI, "Matching is as Easy as Matrix Inversion", *Combinatorica* **7** (1987) 105-113.

Sch80 J.T. SCHWARTZ, "Fast Probabilistic Algorithms for Verification of Polynomial Identities", *J. ACM* **27** (1980) 701- 717.

Vio88 C. VIOLA, Dept. of Mathematics, University of Pisa, private communication.

ANCESTOR TREE FOR ARBITRARY MULTI-TERMINAL CUT FUNCTIONS

by C. K. Cheng and T. C. Hu
Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093

Abstract

In many applications, a function is defined on the cuts of a network. In the max-flow min-cut theorem, the function on a cut is simply the sum of all capacities of edges across the cut; and we want the minimum value of a cut separating a given pair of nodes. To find the minimum cuts separating $\left\lfloor \frac{n}{2} \right\rfloor$ pairs of nodes, we only need n-1 computations to construct the cut-tree.

In general, we can define arbitrary values associated with all cuts in a network, and assume that there is a routine which gives the minimum cut separating a pair of nodes. To find the minimum cuts separating $\left\lfloor \frac{n}{2} \right\rfloor$ pairs of nodes, we also only need n-1 routine calls to construct a binary tree which gives all $\left\lfloor \frac{n}{2} \right\rfloor$ minimum partitions. The binary tree is analogous to the cut-tree of Gomory and Hu.

1. Introduction

Given an undirected network G = (V, E) where V is the set of nodes $\{v_1, v_2, \ldots, v_n\}$ and E is the set of arcs in the network: arc e_{ij} has capacity c_{ij} and connects nodes i and j. A partition of the node set into a subset X and its complement $\overline{X}$ is called a *cut* and is denoted by $(X, \overline{X})$. If $s \in X$ and $t \in \overline{X}$, then the cut $(X, \overline{X})$ *separates* nodes s and t.

We can assign a value to an arbitrary cut $(X, \overline{X})$. In the MAX-FLOW MIN-CUT theorem, the value of a cut $(X, \overline{X})$ is simply the sum $\sum_{i \in X, j \in \overline{X}} c_{ij}$, denoted by $C(X, \overline{X})$, and we wish to find the cut $(X, \overline{X})$ separating s and t which has the minimum value, i.e.,

$$F_{st} = \min_{X} C(X, \overline{X}) \quad \text{with } s \in X \text{ and } t \in \overline{X}. \tag{1}$$

We can view this as minimization of a function.

In other applications [1,6,9,10,11], we may want to define this function F_{st} as, for example,

$$\min_{X} \frac{C(X, \overline{X})}{|X| \cdot |\overline{X}|} \quad \text{with } s \in X \text{ and } t \in \overline{X} \tag{2}$$

where |X| is the cardinality or the size of the set X. The minimum partition based on (2) will divide the network more evenly than the minimum partition based on (1). In VLSI circuit layout, we prefer to have a minimum partition with the number of modules on both sides approximately the same. In other applications, we may need entirely different criteria. Thus, we allow arbitrary values to be associated with any partition and consider the computation for finding the minimum value partition separating a given pair of nodes as a routine call.

In general, we denote the minimum function by $F_{ij}(X, \bar{X})$ with the understanding that $i \in X$ and $j \in \bar{X}$, or we simply use F_{ij} or $F(X, \bar{X})$.

Assume that we wish to determine the $F_{ij}(X, \bar{X})$ values for all pairs of nodes i and j. Our main result is that to find the $\binom{n}{2}$ F_{ij}, we need only n-1 computations of the function F.

To see the difference between arbitrary cut functions and the original minimum cut capacity functions, we first introduce the definition of crossing of cuts.

Two cuts $(X, \bar{X})$ and $(Y, \bar{Y})$ are said to cross each other if each of the following four sets contains at least one node:

$$X \cap Y, \quad X \cap \bar{Y}$$

$$\bar{X} \cap Y, \quad \bar{X} \cap \bar{Y}$$

To find the maximum flows between $\binom{n}{2}$ pairs of nodes in a network, we need n-1 computations where each computation gives a minimum cut. And there exists a set of n-1 minimum cuts which forms a cut-tree [4]. In other words, there exists a set of minimum cuts which do not cross each other.

Here, for arbitrary functions $F_{ij}(X, \bar{X})$, the cuts (X, X) which yield the values for F_{ij} may cross each other (for example, the function defined in (2)). However, we still need only n-1 computations by constructing a binary tree called the ancestor tree.

In Section 2, we prove the existence of the ancestor tree. In Section 3, we give a numerical example. In Section 4 we give the detailed algorithm and its proof. In Section 5, we give some additional remarks including a very simple proof for the Gusfield construction of the Gomory-Hu cut-tree.

2. Ancestor Tree

Let us assume that we have done $\binom{n}{2}$ computations and have found values of F_{ij} between all pairs of nodes in the network. Assume $F_{st}(X, \bar{X})$ is the smallest value among the $\binom{n}{2}$ values. Then for any pair of nodes i and j with $i \in X$ and $j \in \bar{X}$, the cut $(X, \bar{X})$ serves to separate i and j as well; hence we have

$$F_{ij}(X, \bar{X}) = F_{st}(X, \bar{X})$$

Let $F_{ab}(Y, \bar{Y})$ be the smallest value of F where both a and b are in X. Then for any two nodes i and j, where $i \in Y \cap X$ and $j \in \bar{Y} \cap X$, we must have

$$F_{ij} = F_{ab}(Y, \bar{Y})$$

Similarly, let $F_{cd}(Z, \bar{Z})$ have the smallest value with both c and d in $\bar{X}$. Then for any two nodes i and j where $i \in Z \cap \bar{X}$ and $j \in \bar{Z} \cap \bar{X}$, we must have

$$F_{ij} = F_{cd}(Z, \bar{Z})$$

Note that the nodes of the network have been partitioned into four subsets, namely:

$$Y \cap X, \quad \bar{Y} \cap X, \quad Z \cap \bar{X}, \quad \bar{Z} \cap \bar{X}$$

then we know F_{ij} between any two nodes as long as i, j do <u>not</u> belong to the same subset.

We can record the results of the three computations as a binary tree with three internal vertices, as shown in Fig. 1 with $F_{st}(X, \bar{X})$ as the root, $F_{ab}(Y, \bar{Y})$ and $F_{cd}(Z, \bar{Z})$ as the two sons, and the four subsets $Y \cap X$, $\bar{Y} \cap X$, $Z \cap \bar{X}$, $\bar{Z} \cap \bar{X}$ as the four leaves.

The process of partitioning subsets can be continued until each leaf of the tree contains only one node. Then the binary tree has n leaves and n-1 internal vertices; each internal vertex is a computation of the defined function for a given pair of starred nodes. For any two nodes i and j, the lowest common ancestor of their respective leaves gives the partition as well as the value for F_{ij}.

This binary tree is then the Ancestor Tree. Of course, we have proved the existence of the Ancestor Tree by selecting n-1 values from the results of the $\left\lceil \frac{n}{2} \right\rceil$ computations.

In Section 4, we will prove that we can construct the Ancestor Tree by doing only n-1 computations.

3. Numerical Example

Given an n-node network, there are $2^{n-1} - 1$ possible cuts. For each of these cuts, we can arbitrarily assign a value. For a given pair of nodes i and j, there are 2^{n-2} cuts separating i and j and we use $F_{ij}(X, \bar{X})$ to denote the minimum value among the 2^{n-2} cuts. The algorithm for finding F_{ij} for a given pair i and j could be very easy or tedious. We simply consider the algorithm as a routine call. Our main result is to show n - 1 routine calls are sufficient for finding $\left\lceil \frac{n}{2} \right\rceil$ F_{ij}.

Consider the network in Fig. 2 with arc capacities as shown. To fix the idea let us assume that we want to find for all pairs of nodes i and j

$$F_{ij} = \min \frac{C(X,\bar{X})}{|X| \cdot |\bar{X}|} \quad \text{with } i \in X \text{ and } j \in \bar{X}.$$

Assume that we first pick a and b, and find $F_{ab}(X, \bar{X}) = 15/9$. We show the result in Fig. 3. Since the computation is performed for a and b, we call a and b the *seeded* nodes or simply *seeds,* and c, d, e, f are *unseeded* nodes. (*Seeded* nodes are denoted by stars in the Figures.)

We then select a leaf containing one or more unseeded nodes and do a computation between the seeded node and an unseeded node, say between b^* and c. (c then becomes seeded.) The result is shown in Fig. 4. Note that since $F_{bc} < F_{ab}$, the internal vertex $F_{bc}(Y, \bar{Y})$ is put as the root and F_{ab} as its left son. The names of nodes in the leaves have been updated to reflect the three subsets

$$X \cap Y, \quad \bar{X} \cap Y, \quad \bar{Y}$$

In general, let (P,Q) be the new internal vertex just created with $p^* \in P$ and $q^* \in Q$, and F_{ab} (A,B) be the highest ancestor of (P,Q) which satisfies F_{ab} (A,B) > F_{pq} (P,Q). Then we put (P,Q) in the position of (A,B) and attach (A,B) together with its subtree as a son of (P,Q). In the subtree rooted at (A,B), there is a leaf which contains the node p or q, say p. Then we attach the subtree on the P side of the vertex (P,Q). All unseeded nodes, in the leaves (of the subtree rooted at (A,B)) which belong to Q, are now attached as a leaf on the Q side of (P,Q).

The successive computations are shown in Figs. 5, 6, and 7. The process stops when every leaf contains only one seeded node.

4. The Algorithm and Its Proof

Let T_i denote the Ancestor Tree with i leaves. We succcessively build T_i ($i = 1, 2, \cdots, n$).

Initially, the tree T_i consists of one leaf with the names of all nodes in the leaf. Arbitrarily set one node to be the seed.

Ancestor Tree Algorithm

Select a leaf of T_i which contains more than one node and do a computation between the seeded node and an unseeded node in the leaf. This creates a new internal vertex (P,Q) with $p^* \in P$ and $q^* \in Q$ (the unseeded node is now seeded).

(a.) If the newly created vertex F_{pq} has value larger than its father, then F_{pq} remains in its position in T_i and we attach two leaves to F_{pq} to reflect its partitions. One leaf contains the seed p^* and the other leaf contains the seed q^*.

(b.) If F_{pq} has its value less than its father, then we find the lowest ancestor F_{rs} and the highest ancestor F_{ab} which satisfy $F_{rs} \le F_{pq} < F_{ab}$. We put F_{pq} in the previous position of F_{ab} and attach F_{ab} (with its subtree) as a son of F_{pq}. In the subtree with root F_{ab}, there is a leaf which contains the name p or q, say p. Then we attach the subtree on the P side of (P,Q). All unseeded nodes in the subtree which belong to the Q side of the partition are now attached as the leaf on the Q side of (P,Q).

Repeat until there is only one seeded node per leaf in the Ancestor Tree.

Before we prove the Ancestor Tree, we shall prove some Theorems and Lemmas about the arbitrary functions F in a network.

Theorem 1. For any three nodes i, j, and $k \in V$, we have

$$F_{ik} \ge \min(F_{ij}, F_{jk}) \tag{3}$$

Proof: Let F_{ik} have the cut $(X, \bar{X})$ where $i \in X$, $k \in \bar{X}$. Then j either belongs to X or to $\bar{X}$.

If $j \in X$, then the cut $(X, \bar{X})$ serves as a partition separating j and k and we have $F_{ik} \ge F_{jk}$.

If $j \in \bar{X}$, then the cut $(X, \bar{X})$ serves as a partition separating i and j, and we have $F_{ik} \ge F_{ij}$.

In either case, we have $F_{ik} \ge \min(F_{ij}, F_{jk})$.

Theorem 2. Let a, b, $\cdots$, y, z be a sequence of nodes in a network. Then,

$$F_{az} \ge \min(F_{ab}, F_{bc}, \cdots, F_{yz}) \tag{4}$$

Proof: By induction on Theorem 1.

Lemma 1. For any three values F_{ij}, F_{jk}, and F_{ik} between three nodes, two values must be equal and the third value is either greater or the same.

Proof: Putting the smallest value among the three values in the left side of (3), will contradict (3).

Since we know that there are only <u>two</u> distinct values among three values, F_{ij}, F_{jk}, and F_{ik}, we may be able to find the two distinct values by <u>two</u> computations. If we arbitrarily compute

F_{ij} and F_{jk} first and F_{ik} happens to be the largest value, then we need three computations. The following Lemma shows how to avoid this situation.

Lemma 2. If we compute $F_{ij}(X, \bar{X})$ first and find that $j, k \in \bar{X}$, then

$$F_{ik} = \min(F_{ij}, F_{jk})$$

Proof: We know that

$$F_{ik} \geq \min (F_{ij}, F_{jk})$$

so if $F_{ik} \neq \min (F_{ij}, F_{jk})$, then the only possibility is that

$$F_{ik} > \min (F_{ij}, F_{jk})$$

By assumption $j, k \in \bar{X}$ so $(X, \bar{X})$ serves to separate i and k and $F_{ik} \leq F_{ij}$. This is a contradiction.

Lemma 2 is for three nodes. The generalization of Lemma 2 to four or more nodes is not straightforward as seen in the following example.

Let a, b, c, d be four nodes and suppose that we first compute

$$F_{ab}(X, \bar{X}), \text{ and find that b,c belong to } \bar{X}$$

then we compute

$$F_{bc}(Y, \bar{Y}), \text{ and find that c,d belong to } \bar{Y}$$

and then compute F_{cd}. Then it seems that the generalization of Lemma 2 would suggest

$$F_{cd} = \min(F_{ab}, F_{bc}, F_{cd}) \tag{5}$$

Unfortunately (5) is not true as seen from the following example.

Example 2

Using definition (2), we find the cut function in Fig. 8.

$$F_{ab}(X, \bar{X}) < F_{bc}(Y, \bar{Y}), \text{ and } F_{cd} = F(\bar{X}, X)$$

with $X = \{a,d\}$ $Y = \{a,b\}$.

where $F_{ad}(Y,\bar{Y}) > \min (F_{ab}, F_{bc}, F_{cd})$.

The reason for the counterexample to (5) is that $(X, \bar{X})$ and $(Y, \bar{Y})$ cross each other, and the three nodes, a, c, d do not satisfy the conditions of Lemma 2.

Theorem 3. Let a, b, $\cdots$, y, z be any sequence of nodes in a network with

$$\min(F_{ab}, F_{bc}, \cdots, F_{yz}) = F(X,\bar{X})$$

$$\text{and } a \in X, z \in \bar{X},$$

then

$$F_{az} = F(X,\bar{X}) = \min(F_{ab}, F_{bc}, \cdots, F_{yz}) \tag{6}$$

Proof. By assumption $(X, \bar{X})$ separates a and z, so we have

$$F_{az} \leq F(X,\bar{X}) \tag{7}$$

From TH2 and the assumption, we have

$$F_{az} \geq \min(F_{ab}, F_{bc}, \cdots, F_{yz}) = F(X,\bar{X}) \tag{8}$$

The inequalities (7) and (8) imply

$$F_{az} = F(X,\bar{X})$$

Note that Theorem 3 is, in a sense, a generalization of Lemma 2.

Proof. Now we prove the algorithm of the Ancestor Tree by induction on the number of internal vertices in the tree.

In the first stage of construction, we have the internal vertex F_{ab} (A,B) with seed $a \in A$ and seed $b \in B$. We have the following two properties.

(i) For any pairs of nodes i and j *not* in the same leaf F_{ij} is less than or equal to the value of the lowest common anccestor of i and j. (Note that i and j could be seeds or unseeded nodes and there is always one seed per leaf.)

(ii) Let i and j be two seeds and F_{rs} be their lowest common ancestor. Then there exists a sequence of seeds i, a, b, $\cdots$, d, j with every adjacent pair of seeds constituting an internal vertex in the subtree rooted at F_{rs}; for which

$$F_{ij} \geq \min(F_{ia}, F_{ab}, \cdots, F_{dj}) = F_{rs}$$

We shall prove that properties (i) and (ii) are maintained during the successive stages of constructing the Ancestor Tree. Assume that properties (i) and (ii) are true in the kth stage of construction of the Ancestor Tree with k + 1 leaves. Now we pick a leaf containing a seed p and an unseeded node q and create F_{pq} (P,Q).

Let F_{rs} be the lowest ancestor and F_{ab} the highest ancestor of the leaf satisfying

$$F_{rs}(R,S) \leq F_{pq}(P,Q) < F_{ab}(A,B)$$

Without loss of generality, we shall assume that the tree rooted at F_{rs} (R,S) has its left subtree containing nodes in R and F_{ab} (A,B) is the root of the right subtree before F_{pq} is created. And F_{ab} becomes the left subtree of F_{pq} as shown in Fig. 9.

For any pair of nodes i and j with $i \in P \cap S$ and $j \in Q \cap S$, we have $F_{ij} \leq F_{pq}$ sincce (P,Q) provides the cut of the subset S. For any pair of nodes i and j with $i \in R$ and $j \in S$, $F_{ij} \leq F_{rs}$ since the new partition (P,Q) does not replace (R,S) as the lowest common ancestor. So the induction hypothesis (i) still holds.

For the two seeds p and q, $F_{pq} = F_{pq}$ (P,Q) by the routine call and every pair of seeds in an internal vertex is correct by the same reason. Since the algorithm always picks a seed and an unseeded node in the same leaf, a spanning tree connecting seeds is created if every computation is represented by a link connecting two seeds.

This means for any path connecting i and j in the spanning tree, property (ii) holds.

When the algorithm stops there is only one seed per leaf. For any two seeds i and j, we have

$$F_{ij} \leq \min(F_{i1}, F_{12}, \cdots, F_{kj})$$

by hypothesis (i)

$$\text{and } F_{ij} \geq \min(F_{i1}, F_{12}, \cdots, F_{kj})$$

by hypothesis (ii).

Thus we have $F_{ij} = \min(F_{i1}, F_{12}, \cdots, F_{kj})$, where F_{ij} is the lowest common ancestor of i and j.

For every internal vertex, we need to keep the value, the associated partition and the names of the two seeds. Thus we need O (n^2) space. In all the figures, we have not shown the nodes in the associated partitions.

5. General Remarks

Note that the proof of Theorems 1 and 2 and Lemma 1 are exactly the same as that of Gomory and Hu [4]. However, in the original construction [4] of the cut-tree, the algorithm is to pick any two nodes in a supernode and do a maximum flow computation. Here we restrict our selection to a seed and an unseeded node. The construction by Gusfield [5] also has this restriction. Having proved the Ancestor Tree which does not need the property of non-crossing of cuts, we can see more easily why we get stronger results in the cut-tree for MAX flows. In the construction by Gusfield [5], he first constructed a star tree of n nodes with node 1 at the center of the star-tree as shown in Fig. 10.

This is the initial configuration T_1 of cut-tree T. This tree T_1 will be successively modified into $T_2, T_3, \cdots, T_n = T$, where each modification requires one maximum flow computation. We call two nodes connected by a link of the tree T_j neighbors. Thus in T_1, every node j has 1 as the neighbor. In T_1, no value is associated with any link, and we say every node j has an unlabelled neighbor 1. Later, links will have associated values, and we shall call the connected nodes labelled neighbors. In Fig. 10, we declare 1 as a seeded node.

In the tree T_{j-1}, do a maximum flow computation between node j and its unlabelled neighbor i (j becomes a seeded node). The min cut value $C(X, \bar{X})$ is now associated with the link l_{ij} ($i \in X$, $j \in \bar{X}$). All neighbors of i that are in $\bar{X}$ become neighbors of j. (Note all unlabelled neighbors of a seeded node have degree 1 in the tree.) The construction stops after all nodes become seeds.

Note that the Gusfield construction requires n-1 routine calls to a maximum flow minimum cut routine. It is extremely easy to implement (see [5]). The construction is correct even if the routine call provides minimum cuts which cross each other. To see why his construction works, we note the following facts.

Fact 1. At any stage of computation, let a, b, $\cdots$, z, be a sequence of seeded nodes connected by labelled links in the tree T_j, then

$$F_{az} = \min (F_{ab}, F_{bc}, \cdots, F_{yz})$$

Proof: Since each labelled link connecting i and j is a maximum flow computation between i and j, we have the same Theorem 2.

Since each labelled link represents a cut separating a and z we have

$$F_{az} \leq \min (F_{ab}, F_{bc}, \cdots, F_{yz}) \tag{9}$$

Equations (4) and (9) imply $F_{az} = \min (F_{ab}, F_{bc}, \cdots, F_{yz})$ for any sequence of seeded nodes.

Fact 2. Let i and j be unlabelled neighbors in the tree T and j is of degree one, and k is a labelled neighbor of i. (We denote all nodes on the k side of the tree by K, i.e., if the i-k link is removed, all nodes in K are connected to k by a subtree.) Then there exists a minimum cut

$(X, \overline{X})$ separating i and j where the set K is either totally contained in X or totally contained in $\overline{X}$.

Proof: Since every labelled link represents a minimum cut, let $(Y, \overline{Y})$ be the minimum cut corresponding to the i-k link where $i, j \in Y$ and $k \in \overline{Y}$. Since we know there exists a minimum cut $(X, \overline{X})$ separating i and j which does not cross $(Y, \overline{Y})$, then in that cut, K is totally in X or totally in $\overline{X}$.

Comment 1. If we condense all unlabelled neighbors of a seed with the seed, then we have a supernode as in [4].

Comment 2. The n-1 minimum cuts induced by the cut-tree by Gusfield construction may not coincide with the n-1 minimum cuts provided by the routine calls. However, the n-1 cuts represented by the cut-tree are sufficient for finding maximum flows and minimum cuts between all pairs of nodes. The reason is that the routine calls may provide two minimum cuts $(X, \overline{X})$ and $(Y, \overline{Y})$ which cross each other, and the cut-tree will provide two non-crossing minimum cuts $(X, \overline{X})$ and $(Z, \overline{Z})$ which serve the same function as the two crossing cuts (see [4,7,8]).

For minimum partitions separating k nodes ($k \geq 3$), we need $\left[\begin{matrix} n-1 \\ k-1 \end{matrix}\right]$ computations. See Hassin [6].

References

1. C. K. Cheng and T. C. Hu, "Maximum Concurrent Flow and Minimum Ratio Cut," Technical Report CS88-141, Dec. 1988, UCSD, La Jolla, CA 92093.
2. L. R. Ford and D. R. Fulkerson, "Maximal Flow through A Network," *Canadian J. Math.* 8(3), 1956, pp. 399-404.
3. L. R. Ford and D. R. Fulkerson, "Flows in Networks," Princeton University Press, Princeton, NJ, 1962.
4. R. E. Gomory and T. C. Hu, "Multi-terminal Network Flows," *J. of SIAM,* 9(4), Dec. 1961, pp. 551-570.
5. D. Gusfield, "Very Simple Methods for All Pairs Network Flow Analysis," to appear in *SIAM J. on Computing.*
6. R. Hassin, "Solution Bases of Multi-terminal Cut Problems," *Math. of Operations Research,* 13, No. 4, Nov. 1988, pp. 535-542.
7. T. C. Hu, "Integer Programming and Network Flows," Addison-Wesley, 1969.
8. T. C. Hu, "Combinatorial Algorithms," Addison-Wesley, 1982.
9. T. Leighton and S. Rao, "An Approximate Max Flow Min Cut Theorem for Uniform Multi-commodity Flow Problem with Applications to Approximation Algorithm," *IEEE Annual Symposium on Foundations of Computer Science,* 1988, pp. 422-431.
10. D. W. Matula, "Determining Edge Connectivity in O(nm)," *Proceedings of 28th Symposium on Foundations of Computer Science,* 1987, pp. 249-251.
11. F. Shahrokhi and D. W. Matula, "The Maximum Concurrent Flow Problem," Tech. Report, New Mexico, March 1986.

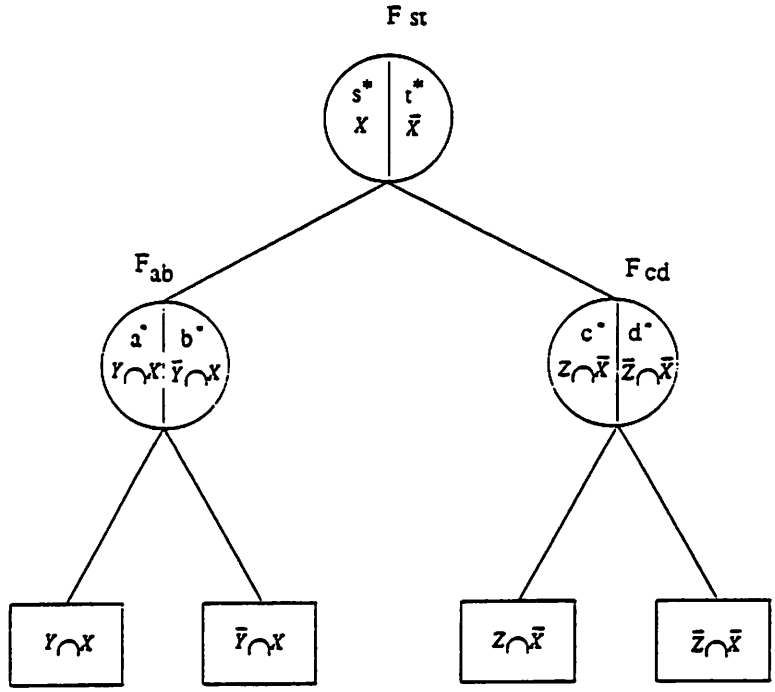


Figure 1

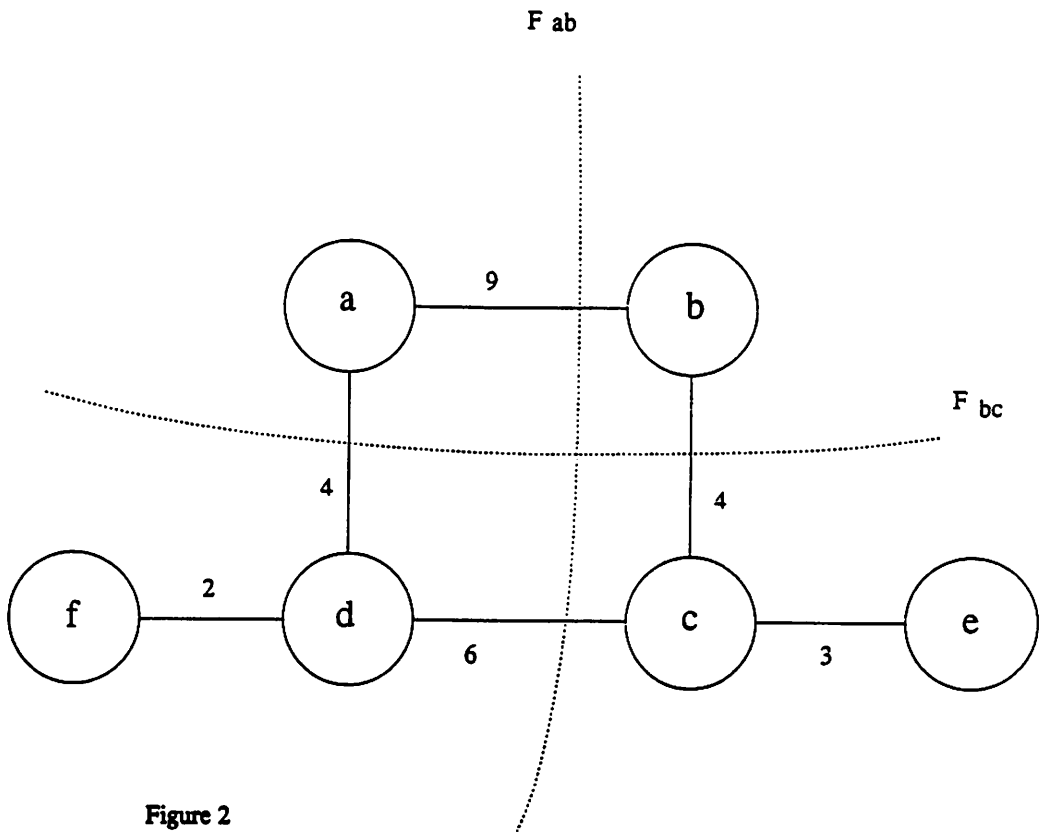


Figure 2

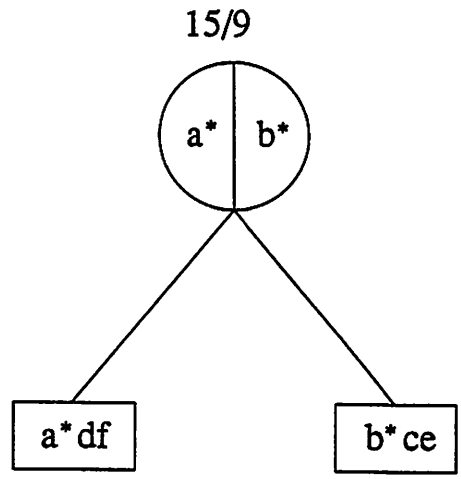


Figure 3

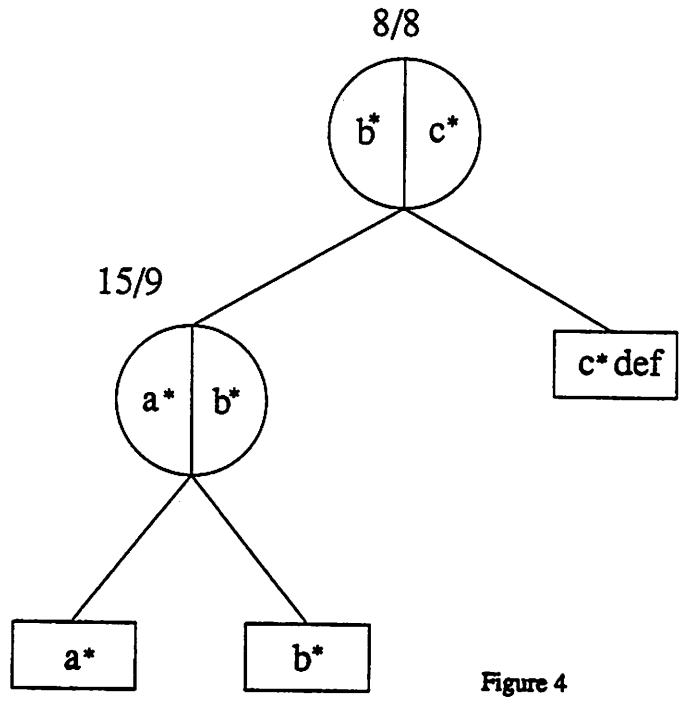


Figure 4

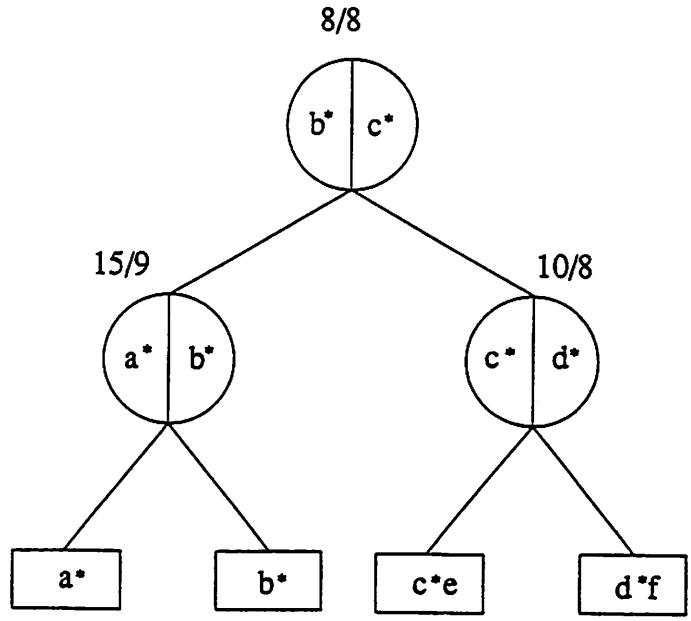


Figure 5

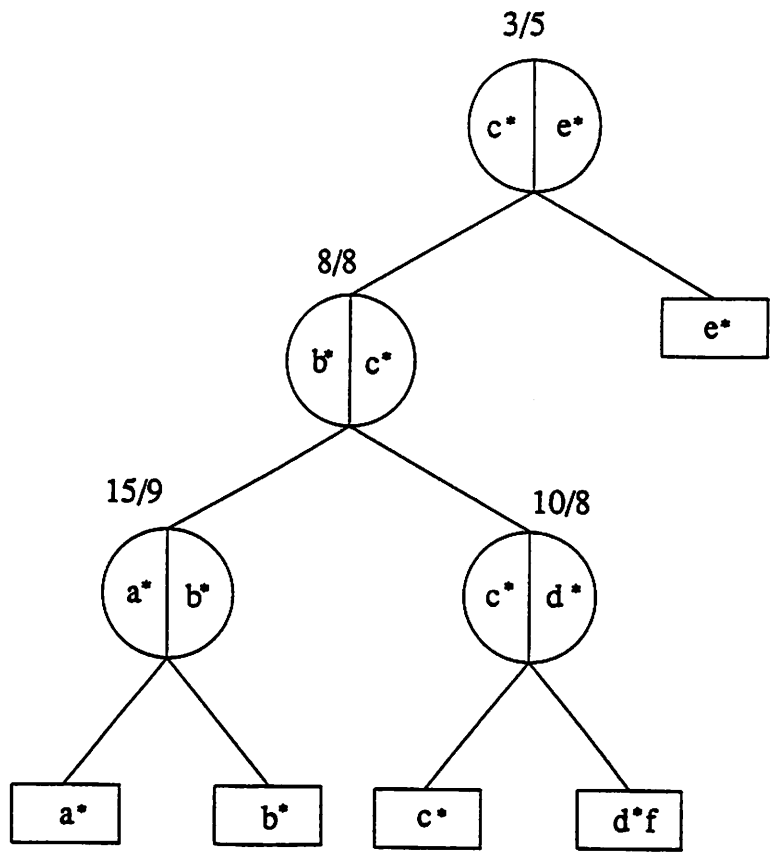


Figure 6

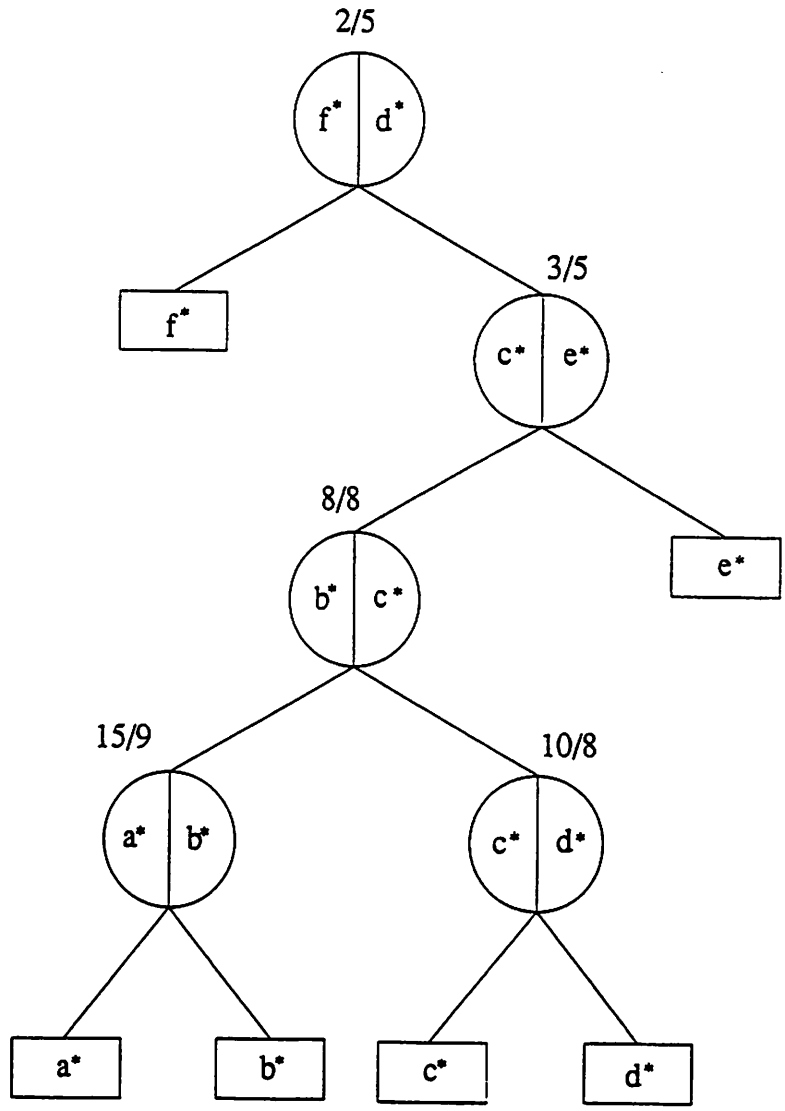


Figure 7

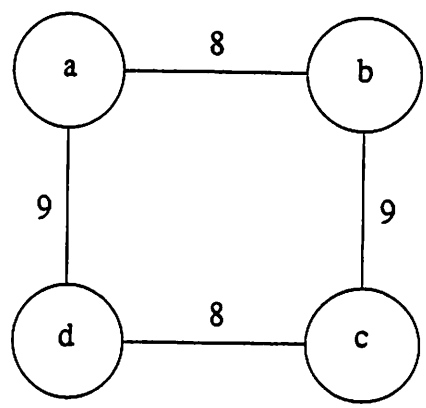


Figure 8

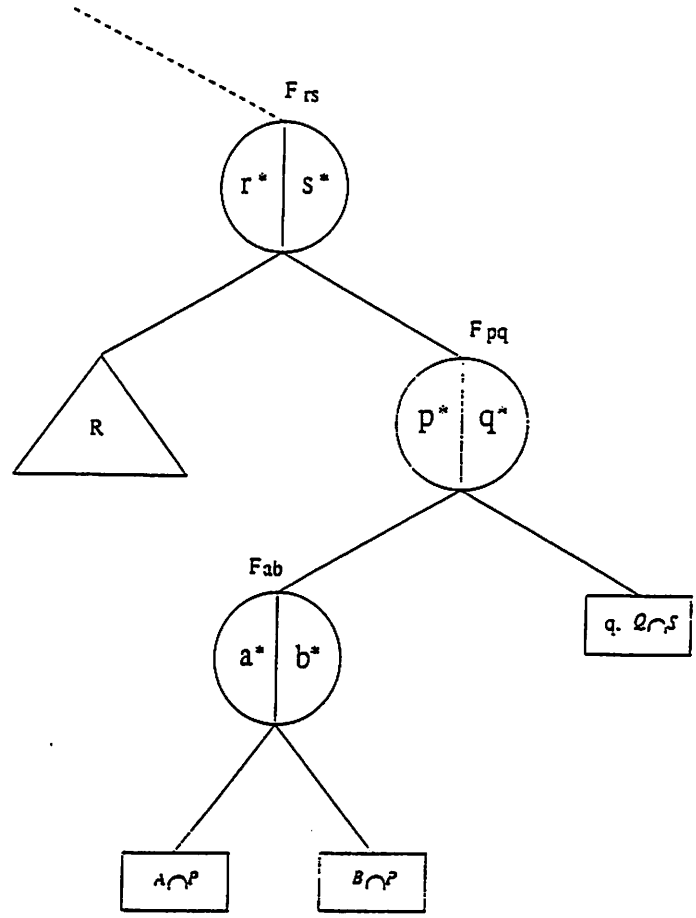


Figure 9

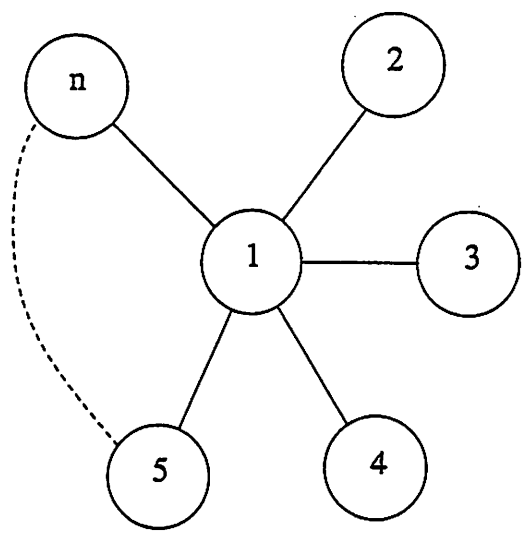


Figure 10

THE GRAPHICAL ASYMMETRIC TRAVELING SALESMAN POLYHEDRON

Sunil CHOPRA

Kellogg Graduate School of Management, Northwestern University, Evanston, IL 60208, U.S.A.

Giovanni RINALDI

Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, 00185 Roma, Italy.

A present trend in the study of the *Symmetric Traveling Salesman Polytope* is to use, as a relaxation of the polytope, the graphical relaxation GTSP. Following a parallel approach for the *Asymmetric Traveling Salesman Polytope*, we define the *Graphical Asymmetric Traveling Salesman Problem* on a general digraph D and its associated polyhedron GATSP(D). We give basic polyhedral results and lifting theorems for GATSP(D) and we give a general condition for a facet defining inequality for GTSP to yield a symmetric facet defining inequality for GATSP. Finally, we describe a family of asymmetric facet defining inequalities for GATSP.

Keywords: Traveling Salesman Problem, directed graph, polyhedron, facet, linear inequality.

1. Introduction

Let $D = (V, A)$ be a directed graph. We denote by $a = (u, v)$ the arc of A having tail u and head v. For every nonempty proper subset W of V, we denote by $\delta_D^+(W)$ and $\delta_D^-(W)$ the sets of arcs defined by $\delta_D^+(W) = \{(u,v) \mid u \in W, v \notin W\}$ and $\delta_D^-(W) = \{(u,v) \mid v \in W, u \notin W\}$, respectively. When W is a singleton node $\{w\}$, we simply write $\delta_D^+(w)$ and $\delta_D^-(w)$. Let $\mathbb{R}^A$ be the space of all real vectors whose components are indexed by A. For x in $\mathbb{R}^A$ we denote by x_a (or $x(u,v)$) the component of x indexed by $a = (u, v)$; for $J \subseteq A$ we denote by $x(J)$ the sum $\sum_{a \in J} x_a$.

A *family of arcs* of D is a collection F of elements of A. Several copies of the same element of A may appear in the collection. For every element a of A, we call *multiplicity* of a in F the number of times a appears in F. A set of arcs of A is a family where every element has multiplicity 1. With every family F of arcs of D we associate a unique incidence vector $\chi^F \in \mathbb{R}^A$ by setting χ_a^F equal to the multiplicity of a in F, for every $a \in A$. Let F_1 and F_2 be two families of arcs of D. We denote by $F_1 + F_2$, $F_1 - F_2$, and kF_1 the families having incidence vectors $\chi^{F_1} + \chi^{F_2}$, $\chi^{F_1} - \chi^{F_2}$, and $k\chi^{F_1}$, respectively.

Let F be a family of arcs of D. By $D'[F]$ we denote the support digraph of F, i.e. the subgraph obtained from D by removing the arcs not contained in F. By $D[T]$ we denote the directed multigraph obtained by replicating every arc a as many times as the multiplicity of a in F.

Let $G = (V, W)$ be an undirected graph. We denote by $e = [u, v]$ the edge of E with endpoints u and v, and for $x \in \mathbb{R}^E$ we denote by x_e (or $x(u,v)$) the component of x indexed by e. For every nonempty proper subset W of V, we denote by $\delta_G(W)$ the set of edges defined by $\delta_G(W) = \{[u,v] \mid u \in W, v \notin W\}$, and if W is a singleton node $\{w\}$ we simply write $\delta_G(w)$. All the above definitions and notations on the families of arcs apply also to the *families of edges*, when the directed graph D and its arc set A are replaced with the undirected graph G and its edge set E, respectively.

A *tour* of D is a family T of arcs of D such that:

(a) $|\delta_{D[T]}^+(u)| = |\delta_{D[T]}^-(u)|$ for all u in V;

(b) $D[T]$ is connected.

An *undirected tour* of G is a family Θ of edges of G such that:

(a') $|\delta_{G[\Theta]}(u)|$ is even for all u in V;
(b') $G[\Theta]$ is connected.

In the following we use a roman or a greek letter to denote a tour or an undirected tour, respectively. For the sake of simplicity we use the term "tour" also in the undirected case, every time this does not generate any ambiguity.

A tour T is *minimal* if no proper subfamily of T is a tour, i.e. if the removal of any directed cycle from $D[T]$ produces a disconnected multigraph. A tour of D is *extremal* if its incidence vector is not a convex combination of other tours of D. An extremal tour is necessarily minimal, but the converse is not necessarily true (see Section 2). An *equality tour* T for an inequality $cx \geq c_0$ of $\mathbb{R}^A$ is a tour with $c\chi^T = c_0$. The sets of all tours and of all extremal tours of D are denoted by $\mathcal{T}_D^*$, and $\mathcal{T}_D$, respectively.

The definitions and notations of the previous paragraph apply, *mutatis mutandis*, to the undirected tours.

Given arc weights $w_a \in \mathbb{R}$, the *Graphical Asymmetric Traveling Salesman Problem* (*gatsp*) is to find a minimum weight tour in $\mathcal{T}_D^*$. The *Graphical Asymmetric Traveling Salesman Polyhedron* associated with D (GATSP(D)) is the convex hull of the incidence vectors of the elements of $\mathcal{T}_D^*$, i.e.:

$$\text{GATSP}(D) = \text{conv}\left\{\chi^T \mid T \in \mathcal{T}_D^*\right\}.$$

A *directed Hamiltonian cycle* of D is a tour H with $|\delta^+_{D[H]}(u)| = 1$ for every u in V. Given a complete directed graph D_n, the *Asymmertic Traveling Salesman Problem* (*atsp*) is to find a minimum weight Hamiltonian cycle in D_n. We denote by $\mathcal{H}_n$ the set of all directed Hamiltonian cycles of D_n. The *Asymmetric Traveling Salesman Polytope* associated with D_n (ATSP(n)) is the convex hull of the incidence vectors of the elements of $\mathcal{H}_n$, i.e.:

$$\text{ATSP}(n) = \text{conv}\left\{\chi^H \mid H \in \mathcal{H}_n\right\}.$$

Given edge weights $w_e \in \mathbb{R}$, the *Graphical Traveling Salesman Problem* (*gtsp*) is to find a minimum weight tour in $\mathcal{T}_G^*$. The *Graphical Traveling Salesman Polyhedron* associated with G (GTSP(G)) is the convex hull of the incidence vectors of the elements of $\mathcal{T}_G^*$, i.e.:

$$\text{GTSP}(G) = \text{conv}\left\{\chi^\Theta \mid \Theta \in \mathcal{T}_G^*\right\}.$$

We refer to [4] for the necessary background in polyhedral theory and for a survey on the polyhedral aspects of the *traveling salesman problem*.

The counterpart of *gatsp* for undirected graphs (*gtsp*) has been studied in [2], [3], [5], and [8]. To the best of our knowledge *gatsp* has not been investigated yet. The advantages in studying *gatsp* as a relaxation to *atsp* are the same as those mentioned in [2] and [3] for *gtsp*. In particular

(a) if the objective function coefficients satisfy the triangular inequality (like in most of the real-world applications), *gatsp* has always a directed Hamiltonian cycle among the optimal solutions, and so *atsp* can be solved as a *gatsp*;
(b) in some applications the requirement that a tour visit a node only once may not be necessary, and so *gatsp* may be a more appropriate formulation for the problem;
(c) *gatsp* is defined for general (even not Hamiltonian) graphs, and for these graphs it is not necessary to add artificial arcs, as for the case when to formulate the problem *atsp* is used instead;

(d) several facets of GATSP are defined for very sparse graphs and the coefficients of the missing edges can be computed by a sequential lifting procedure; often these coefficients can be given in closed form (see Remark 3.1);
(e) it is often possible to extend results on the polyhedral structure of GATSPto obtain results on ATSP, like has been done for the undirected case (see [6] and [7]);
(f) several results on the polyhedral structure of GTSP can be extended to GATSP (see Section 3). An unnecessary duplication of work can be saved in this way in the proofs.

In Section 2 we give general results on GATSP, we define its basic facet defining inequalities and give some general lifting theorems. In Section 3 we give conditions under which a facet defining inequality for GTSP yields a facet defining inequality for GATSP. We use these conditions to show that several families of symmetric inequalities define facets of GATSP. In Section 4 we describe some families of asymmetric facet defining inequalities for GATSP. Unlike the inequalitites of Section 3, these inequalities do not have a counterpart in the undirected case.

For space reasons all proofs have been omitted. The interested reader can find them in [1].

2. Basic properties of GATSP(D)

If a directed graph D is not strongly connected, then GATSP(D) is empty (if T is a tour, the subgraph $D'[T]$ of D is strongly connected). On the other hand if D is strongly connected, clearly one can construct a tour in D.

For every tour T of D, $\chi^T(\delta^+_{D[T]}(u)) - \chi^T(\delta^-_{D[T]}(u)) = 0$, for all u in V, and $\chi^T(\delta^+_{D[T]}(W)) + \chi^T(\delta^-_{D[T]}(W)) \geq 2$, for every nonempty proper subset W of V. It follows that GATSP(D) $\subseteq \{x \in \mathbb{R}^A \mid F_D x = 0\}$, where F_D is the node-arc incidence matrix of D. Hence GATSP(D) is not a full dimensional polyhedron.

Theorem 2.1. *If D is strongly connected the dimension of* GATSP(D) *is* $|A| - |V| + 1$.

Since GATSP is not full dimensional, two inequalities $\pi' x \geq \pi'_0$ and $\pi'' x \geq \pi''_0$ are ***equivalent***, i.e., define the same facet of the polyhedron, if and only if there exist $\xi > 0$ and $\lambda \in \mathbb{R}^{V'}$, such that $\pi' = \xi\pi'' + \lambda F'_D$ and $\pi'_0 = \xi\pi''_0$. Here V' denotes the node set $V - \{\overline{v}\}$ for some $\overline{v} \in V$, and F'_D denotes the submatrix of F_D obtained on taking the rows corresponding to V'. Consequently, recognizing when two facet defining inequalities for GATSP are equivalent is not as easy as in the case of a full dimensional polyhedron. To overcome this problem we use the following definition of *standard* form for an inequality.

An inequality $\pi x \geq \pi_0$ facet defining for GATSP is in ***standard form*** if $F_D \pi = 0$. It follows that two inequalities in standard form defining facets of GATSP are equivalent if and only if one is obtained on multiplying the other by a nonzero number. In the case of a complete digraph, for any facet defining inequality $\pi x \geq \pi_0$ an equivalent inequality $(\pi + \lambda F'_{D_n})x \geq \pi_0$ can be obtained by defining

$$\lambda_v = \frac{1}{2n}\sum_{u \neq v}(\pi(v,u) - \pi(u,v)) - \frac{1}{2n}\sum_{u \neq \overline{v}}(\pi(\overline{v},u) - \pi(u,\overline{v})) \quad \text{for } v \in V'. \tag{2.1}$$

A facet defining inequality $\pi x \geq \pi_0$ for GATSP(D_n) is said to be ***symmetric*** if

$$\pi(u,v) = \pi(v,u) \quad \text{for every } (u,v) \in A. \tag{2.2}$$

Observe that a symmetric inequality is always in standard form. It follows that to check whether an inequality is equivalent to a symmetric one, it is sufficient to find an inequality in standard form

equivalent to it. A facet defining inequality for GATSP(D_n) is called *asymmetric* if any inequality in standard form equivalent to it does not satisfy (2.2).

In order to study the facets of a polyhedron it is interesting to know the dimension of the convex hull of its extreme points. If this dimension coincides with the dimension of the polyhedron, several facets (those that really matter in a polyhedral cutting-plane algorithm) are the convex hull of extreme points of the polyhedron. Otherwise, to describe any facet of the polyhedron, except at most one, some extreme rays have to considered.

It is difficult to characterize the extreme points of GATSP(D), for a general digraph D. Every extreme tour has to be minimal, but the converse does not hold. See, for example, Figure 2.1(a), where a minimal tour is shown that can be expressed as a convex combination of the tours of the figures 2.1(b) and 2.1(c).

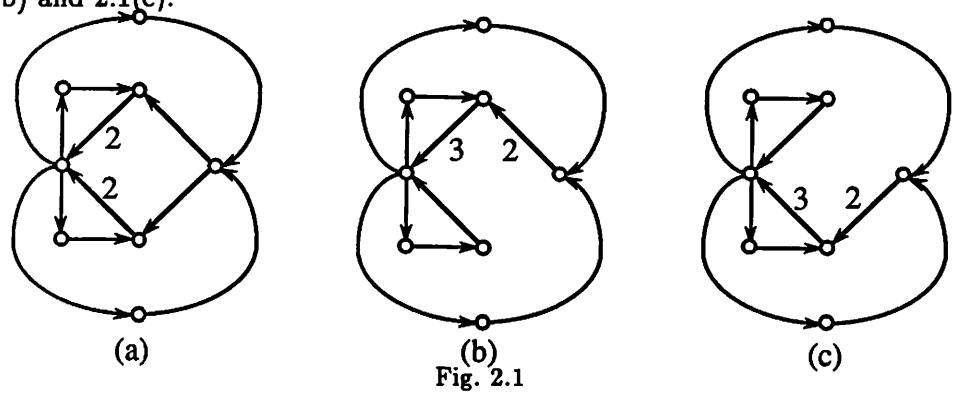


Fig. 2.1

Also the dimension of the convex hull of the extreme points of GATSP(D) seems hard to identify in general. As an example consider the graphs D^1 and D^2 of Figure 2.2, which differ only by one arc. The dimension is zero for the first graph and two for the second.

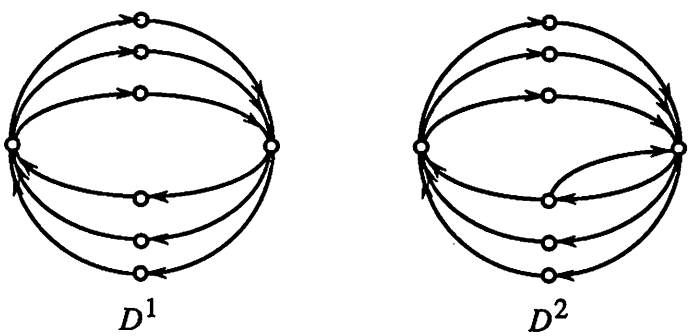


Fig. 2.2

When D is a *bidirected* graph, that is (u,v) in A implies (v,u) in A, the dimension depends upon the number of bridges. A *bridge* is a pair (u,v), (v,u) whose removal disconnects D. In the following we study facet defining inequalities of GATSP(D), where D has a bidirected graph as a subgraph, and so the case of a bidirected graph is of particular interest for us.

Theorem 2.2. *If D is bidirected and has k bridges, the convex hull of the extreme points of* GATSP(D) *has dimension* $|A| - |V| - k + 1$.

The integer points of the polyhedron defined by

$$F_D x = 0, \tag{2.3}$$

$$x(\delta_D^+(W)) + x(\delta_D^-(W)) \geq 2 \quad \text{for all } \emptyset \neq W \subset V, \tag{2.4}$$

$$x \geq 0, \tag{2.5}$$

are the incidence vectors of all tours of D. We call (2.3) the *flow equations*, (2.4) the *cut inequalities* and (2.5) the *trivial inequalities*. Observe that while in the directed case (2.3), (2.4) and (2.5) provide a true integer linear programming formulation of *gatsp*, for the undirected case the condition that every node has an even degree in every tour cannot be stated in a simple form. A simple linear transformation can be used to derive an integer linear programming formulation for *gtsp* from (2.3), (2.4), and (2.5).

Theorem 2.3. *For $a \in A$ the trivial inequality $x_a \geq 0$ is facet defining for* GATSP(D) *if and only if the subgraph of D obtained on deleting a is strongly connected.*

Theorem 2.4. *For $\emptyset \neq W \subset V$, the cut inequality $x(\delta_D^+(W)) + x(\delta_D^-(W)) \geq 2$ is facet defining for* GATSP(D) *if and only if both the subgraphs of D induced by W and by $V - W$, respectively, are strongly connected.*

An inequality $\pi x \geq \pi_0$ in $\mathbb{R}^A$ satisfies the *shortest path condition* if for every (u, v) in A and for every directed path P_{uv} connecting u to v, $\pi(u, v) \leq \pi(P_{uv})$.

Proposition 2.5. *A facet defining inequality for* GATSP(D) *either is a trivial inequality or satisfies the shortest path condition.*

The next two lemmata give liftings for facets of GATSP(D).

Lemma 2.6. *Let $\pi' x' \geq \pi_0$ be a valid inequality for* GATSP(D') *such that D' has a directed cycle C with a chord $\bar{a} = (u, v)$ and $\pi'(C) = 0$. Form D from D' by deleting $\bar{a}$. The inequality $\pi' x' \geq \pi_0$ is facet defining for* GATSP(D') *if it satisfies the shortest path property and $\pi x \geq \pi_0$ is facet defining for* GATSP(D), *where π is the restriction of π' to the arcs of D.*

Consider a directed graph D^* with a directed chordless cycle C. Shrink the cycle C by contracting all arcs in C and deleting parallel arcs to get $D[C]$.

Lemma 2.7. *Let $\pi^* x^* \geq \pi_0$ be a valid inequality for* GATSP(D^*) *satisfying the shortest path property and $\pi_a^* = 0$, for $a \in C$. The inequality $\pi^* x^* \geq \pi_0$ is facet defining for* GATSP(D^*) *if $\pi x \geq \pi_0$ is facet defining for* GATSP($D^*[C]$), *where π is the restriction of π^* to the arcs of $D^*[C]$.*

Note that every facet defining inequality $\pi x \geq \pi_0$ with $\pi(C) = 0$ can be converted to an equivalent inequality $\pi' x \geq \pi_0$ (by adding to it a suitable linear combination of the flow equations) such that $\pi'_a = 0$, for all a in C.

Let $\pi x \geq \pi_0$ be a facet defining inequality for GATSP(D), with $D = (V, A)$. Let $D' = (V', A')$ be a strongly connected digraph. Let $D^* = (V^*, A^*)$ be the digraph defined by

$$\begin{aligned} V^* &= V - \{\bar{v}\} + V', \\ A^* &= A - \delta_D^+(\bar{v}) - \delta_D^-(\bar{v}) + A' + \left\{(v, u) \mid (v, \bar{v}) \in A, u \in V_v^- \subseteq V'\right\} \\ &\quad + \left\{(u, v) \mid (\bar{v}, v) \in A, u \in V_v^+ \subseteq V'\right\}, \end{aligned}$$

where $\bar{v}$ is a node of V and V_v^- and V_v^+ are subsets of V' associated with v, for $v \in V - \{\bar{v}\}$. The inequality $\pi^* x^* \geq \pi_0$ defined by

$$\pi^*(u, v) = \begin{cases} \pi(u, v) & \text{for } (u, v) \in A - \delta_D^+(\bar{v}) - \delta_D^-(\bar{v}), \\ 0 & \text{for } (u, v) \in A', \\ \pi(u, \bar{v}) & \text{for } (u, v) \in \delta_{D^*}^-(V'), \\ \pi(\bar{v}, u) & \text{for } (u, v) \in \delta_{D^*}^+(V'), \end{cases}$$

is said to obtained from $\pi x \geq \pi_0$ by *zero lifting* of the node $\bar{v}$.

Theorem 2.8. *If $\pi^* x^* \geq \pi_0$ is obtained by zero lifting of a facet defining inequality $\pi x \geq \pi_0$ for* GATSP(D), *then $\pi^* x^* \geq \pi_0$ is facet defining for* GATSP(D^*).

3. Facets of GATSP from those of GTSP

Consider an undirected complete graph with n nodes $K_n = (V_n, E_n)$ and the associated polyhedron GTSP(K_n). Cornuéjols et al. [2], and Naddef and Rinaldi [5], [8] have studied this polyhedron and given several families of facet defining inequalities for it.

Assume that the inequality

$$\mu y \geq \mu_0 \tag{3.1}$$

is facet defining for GTSP(K_n). A subgraph $G = (V_n, E)$ of K_n is said to be a *skeleton* for (3.1) if the restriction of (3.1) to $\mathbb{R}^E$ defines a facet of GTSP(G). The inequality (3.1) is said to be *stable for the skeleton* G if for every edge $e \in E_n - E$ there exists an equality tour Θ_e for (3.1) containing e and only edges in E. This is equivalent to saying that the sequential lifting coefficients of the edges in $E_n - E$ do not depend on the lifting sequence (for a definition of sequential lifting see, e.g., [4]). From this definition it follows that if (3.1) is stable for the skeleton G, then for every graph $G' = (V_n, E')$ with $E \subseteq E'$, the restriction of (3.1) to $\mathbb{R}^{E'}$ defines a facet of GTSP(G'). The definitions of *skeleton* and of *inequality stable for a skeleton* apply, *mutatis mutandis*, also to the case of a directed graph.

Remark 3.1. All the facet defining inequalities for GTSP(K_n) described in [2], [5], and [8] have the property that they are stable for a skeleton which is quite sparse. This nice property simplifies the analysis of this inequalities and the proofs that they define facets of the polyhedron. For the inequalities described in this section and in the following one this property holds as well.

Let $G = (V, E)$ be a skeleton of (3.1) and let $D = (V, A)$ be the directed graph obtained from G on replacing each edge $e = [u_i, u_j]$ by the pair of arcs (u_i, u_j) and (u_j, u_i). Here we give a set of sufficient conditions under which a facet defining inequality for GTSP(G) gives a facet defining inequality with symmetric coefficients for GATSP(D).

Define the inequality

$$\pi x \geq \pi_0 \tag{3.2}$$

where $\pi(u_i, u_j) = \pi(u_j, u_i) = \mu_e$, where $e = [u_i, u_j]$, and $\pi_0 = \mu_0$. For any tour T of D_n one can form an undirected tour Θ of K_n where the multiplicity of any edge $e = [u_i, u_j]$ in Θ is the sum of the multiplicity of the arcs (u_i, u_j) and (u_j, u_i) in T. If T violates (3.2) then Θ violates (3.1). Thus the inequality (3.2) is valid for GATSP(D_n).

Let S be any spanning tree of G with edges $E(S)$. Consider the set of arcs

$$S_D = \{(u_i, u_j), (u_j, u_i) \mid [u_i, u_j] \in E(S)\}.$$

Theorem 3.1. *If the equality tours for the inequality (3.1) satisfy the condition (3.3) stated below, then D is a skeleton for the inequality (3.2). Moreover, if (3.1) is stable for G so is (3.2) for D.*

(3.3) *There exists a spanning tree S and an ordering of the edges in $E - E(S)$, say $\langle e_1, e_2, \ldots, e_t \rangle$ where $t = |E - E(S)|$, such that for each edge e_i there exists an equality tour $\Theta(e_i)$ for (3.1) with edges in E and multiplicity of e_i equal to one, and a cycle $C(e_i)$ in $\Theta(e_i)$ that contains e_i with $C(e_i) \subseteq E(S) \cup \{e_1, e_2, \ldots, e_i\}$.*

Now we use the above result to prove that several families of facet defining inequalities for GTSP give facet defining inequalities for GATSP.

3.1. The s-path inequalities

We begin with the k-path configurations described in [2] for the undirected case. For any odd $k \geq 3$ and any k-tuple of positive integers $(n_1, \ldots, n_k)$, with $n_i \geq 2$ for $i \in \{1, \ldots, k\}$, let $P(n_1, \ldots, n_k)$ be the directed graph on the node set $V_P = \{Y, Z\} \cup \{u_j^i \mid i \in \{1, \ldots, k\};\ j \in \{1, \ldots, n_i\}\}$. $P(n_1, \ldots, n_k)$ is a *directed k-path configuration*. For convenience label $u_0^i = Y$, and $u_{n_i}^i = Z$. The arc set is given by

$$A_P = \{(u_j^i, u_{j+1}^i), (u_{j+1}^i, u_j^i) \mid i \in \{1, \ldots, k\};\ j \in \{0, \ldots, n_i\}\}.$$

The directed 3-path configuration $P(2, 2, 2)$ is as shown in Figure 3.1.

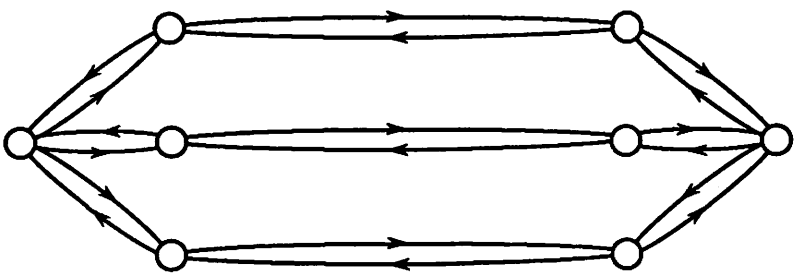


Fig. 3.1

Let $D = (V_P, A)$ be any graph that contains $P(n_1, \ldots, n_k)$ as a subgraph. A *s-path inequality* associated with $P(n_1, \ldots, n_k)$ is the inequality

$$cx \geq c_0 \tag{3.1.1}$$

on $\mathbb{R}^A$, where c and c_0 are defined as follows:

$$c_0 = 1 + \sum_{i=1}^{k} \frac{n_i + 1}{n_i - 1},$$

$$c(Y, Z) = c(Z, Y) = 1,$$

$$c(u_j^i, u_q^i) = \frac{|j - q|}{n_i - 1} \quad \text{for all } i \in \{1, \ldots, k\}; j \neq q \in \{0, \ldots, n_i + 1\},$$

$$c(u_j^i, u_q^r) = \frac{1}{n_i - 1} + \frac{1}{n_r - 1} + \left| \frac{j - 1}{n_i - 1} - \frac{q - 1}{n_r - 1} \right|$$
$$\text{for all } i \neq r \in \{1, \ldots, k\};\ j \in \{1, \ldots, n_i\};\ q \in \{1, \ldots, n_r\}.$$

Note that the coefficients of (3.1.1) described above are symmetric and identical to those described in [2] for the path inequality on the corresponding undirected graph $G = (V_P, E)$.

Theorem 3.1.1. *The s-path inequality is facet defining for* GATSP(D).

3.2. The s-wheelbarrow inequalities

Next we turn to the k-wheelbarrow configuration also discussed in [2] for the undirected case. For any odd integer $k \geq 3$ and k-tuple of positive integers $(n_1, n_2, \ldots, n_k)$ with $n_i \geq 2$ for $i \in \{1, 2, \ldots, k\}$ we define a *directed k-wheelbarrow configuration* $W(n_1, \ldots, n_k) = (V_W, A_W)$ where the node set V_W is given by

$$V_W = V_P - \{Z\}$$

and arc set A_W is given by

$$A_W = A_P - \left\{(u^i_{n_i}, Z), (Z, u^i_{n_i}) \mid i \in \{1, \ldots, k\}\right\} + \left\{(u^i_{n_i}, u^{i+1}_{n_{i+1}}), (u^{i+1}_{n_{i+1}}, u^i_{n_i}) \mid i \in \{1, \ldots, k\}\right\}.$$

All superscript indices are taken modulo k where $0 = k$. $W(2, 2, 2)$ is as shown in Figure 3.2.

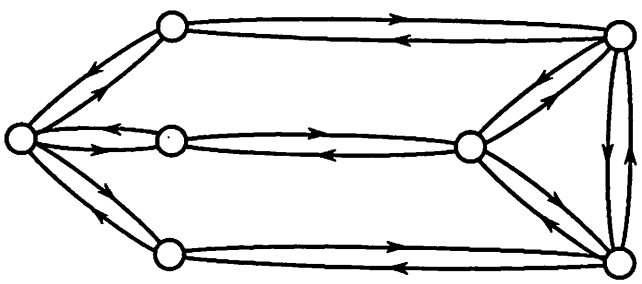


Fig. 3.2

Let $D = (V_W, A)$ be any directed graph that contains $W(n_1, \ldots, n_k)$ as a subgraph. A *s-wheelbarrow inequality* associated with $W(n_1, \ldots, n_k)$ is the inequality

$$cx \geq c_0 \tag{3.2.1}$$

on $\mathbb{R}^A$, where c and c_0 are defined as for the s-path inequality (3.1.1). The inequality (3.2.1) is a restriction of the s-path inequality (3.1.1). The coefficients of (3.2.1) are symmetric and identical to those described in [2] for the wheelbarrow inequality on the corresponding undirected graph $G = (V_W, E)$.

Theorem 3.2.1. *The s-wheelbarrow inequality (3.2.1) is facet defining for* GATSP(D).

3.3. The s-bycicle inequalities

Next we turn to the k-bicycle configuration also discussed in [2] for the undirected case. For any odd integer $k \geq 3$ and k-tuple of positive integers $(n_1, n_2, \ldots, n_k)$ with $n_i \geq 2$ for $i \in \{1, 2, \ldots, k\}$ we define a *directed k-bicycle configuration* $B(n_1, \ldots, n_k) = (V_B, A_B)$ where the node set

$$V_B = V_W - \{Y\},$$

and arc set

$$A_B = A_W - \left\{(u^i_1, Y), (Y, u^i_1) \mid i \in \{1, \ldots, k\}\right\} + \left\{(u^i_1, u^{i+1}_1), (u^{i+1}_1, u^i_1) \mid i \in \{1, \ldots, k\}\right\} + \left\{(u^i_1, u^{i+2}_1), (u^{i+2}_1, u^i_1), (u^i_{n_i}, u^{i+2}_{n_{i+2}}), (u^{i+2}_{n_{i+2}}, u^i_{n_i}) \mid i \in \{1, \ldots, k\}\right\}.$$

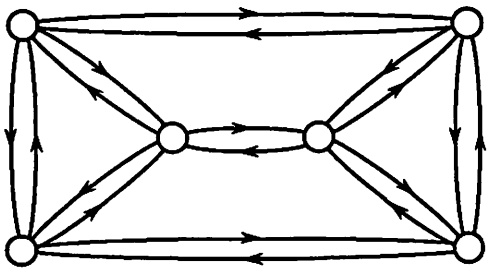


Fig. 3.3

All superscript indices are taken modulo k with $0 = k$. $B(2,2,2)$ is as shown in Figure 3.3.

Let $D = (V_B, A)$ be any graph that contains $B(n_1, \ldots, n_k)$ as a subgraph. A *s-bicycle inequality* associated with $B(n_1, \ldots, n_k)$ is the inequality

$$cx \geq c_0 \tag{3.3.1}$$

on $\mathbb{R}^A$, where c and c_0 are as defined for the s-path inequality (3.1.1). The inequality (3.3.1) is a restriction of (3.1.1). The coefficients of (3.3.1) are symmetric and identical to those described in [2] for the bicycle inequality of the corresponding undirected graph $G = (V_B, E)$.

Theorem 3.1.6. *The s-bicycle inequality (3.3.1) is facet defining for* GATSP(D).

3.4. The s-crown inequalities

Now we turn to the *crown inequalities* described in [8]. For any integer $k \geq 2$ let $CR(p) = (V_C, A_C)$ be a *directed crown configuration*, where $p = 4k$ and

$$\begin{aligned} V_C &= \{u_i \mid i \in \{1, \ldots, p\}\}, \\ A_C &= \{(u_i, u_{i+1}), (u_{i+1}, u_i), (u_i, u_{2k+i}), \\ &\qquad (u_{2k+i}, u_i), (u_i, u_{i+2}), (u_{i+2}, u_i) \mid i \in \{1, \ldots, p\}\}. \end{aligned}$$

All indices are modulo p with $0 = p$. The arcs entering and leaving u_1 in $CR(8)$ are as shown in Figure 3.4.

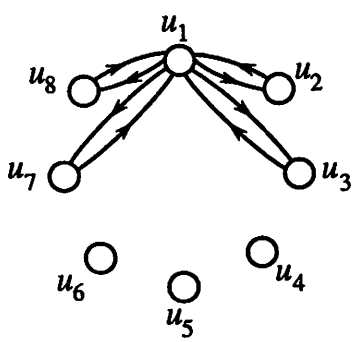


Fig. 3.4

Let $D = (V_C, A)$ be any directed graph that contains $CR(p)$ as a subgraph. A *s-crown inequality* associated with $CR(p)$ is the inequality

$$cx \geq c_0, \tag{3.4.1}$$

where $c_0 = 12k(k-1) - 2$, and

$$c(u_i, u_{i+j}) = \begin{cases} 4k - 6 + |j| & \text{for } 1 \le |j| \le 2k-1, \\ 2(k-1) & \text{for } j = 2k. \end{cases}$$

The coefficients of (3.4.1) are symmetric and identical to those described in [8] for the crown inequality on the corresponding undirected graph $G = (V_C, E)$.

Theorem 3.1.8. *The s-crown inequality (3.4.1) is facet defining for* GATSP(D).

3.5. Composition of symmetric inequalities

Let $G^1 = (V_{n_1}, E_1)$ and $G^2 = (V_{n_2}, E_2)$ be the skeletons of the inequalities

$$\mu^1 y^1 \ge \mu_0^1, \tag{3.5.1}$$

$$\mu^2 y^2 \ge \mu_0^2, \tag{3.5.2}$$

facet defining for GTSP(K_{n_1}) and GTSP(K_{n_2}), respectively. Let $V^1 = \{v_1^1, v_2^1, \ldots, v_s^1\}$ and $V^2 = \{v_1^2, v_2^2, \ldots, v_s^2\}$, with $s \ge 2$, be subsets of V_{n_1} and V_{n_2}, respectively. For simplicity we assume that the subgraph of G^1 induced by V^1 is isomorphic to the subgraph of G^2 induced by V^2, and that v_i^1 corresponds to v_i^2, for $i = 1, 2, \ldots, s$, in the isomorphism.

The *s-sum* of G^1 and G^2 is the graph $G = (V_n, E)$, obtained on identifying the node v_i^1 with v_i^2, for $i = 1, 2, \ldots, s$, and removing the resulting parallel edges. Here $n = n_1 + n_2 - s$, $V_n = (V_{n_1} - V^1) + (V_{n_2} - V^2) + W$, $W = \{w_1, w_2, \ldots, w_s\}$, w_i results from the identification of v_i^1 and v_i^2, for $i \in \{1, 2, \ldots, s\}$, and finally E is such that the subgraph of G induced by $(V_{n_i} - V^i) + W$ is isomorphic to G^i, for $i = 1, 2$.

If the condition

$$\mu^1(v_i^1, v_j^1) = \mu^2(v_i^2, v_j^2) \quad \text{for all } i \ne j \in \{1, 2, \ldots, s\} \tag{3.5.3}$$

is satisfied (for $s = 2$ this can always be obtained by multiplying (3.5.2) by a suitable multiplier), we can define an inequality

$$\mu y \ge \mu_0 \tag{3.5.4}$$

on $\mathbb{R}^{E_n}$ *obtained by s-sum of* (3.5.1) *and* (3.5.2) in the following way. To the edges with one endpoint in $V_1 - W$ and the other in $V_2 - W$, that we call the *crossing edges* of (3.5.4), we assign an ordering

$$\langle e_1, e_2, \ldots, e_r \rangle. \tag{3.5.5}$$

Then

$$\mu(v_i, v_j) = \mu^r(v_i, v_j) \quad \text{for } \{v_i, v_j\} \subseteq V^r \text{ and } r \in \{1, 2\}.$$

For a crossing edge e the coefficient μ_e is computed by a sequential lifting procedure based on the ordering (3.5.5). The right hand side μ_0 is the length of an optimal solution of *gtsp* in G when the weights are given by μ.

If (3.5.3) and

$$\mu^1(v_i^1, v_{i+j}^1) = \sum_{p=1}^{j-1} \mu^1(v_p^1, v_{p+1}^1) \quad \text{for all } j \in \{1, 2, \ldots, s\}$$

hold (for $s = 2$ this is clearly always the case), the inequality (3.5.4) is said to be obtained by a *linear s-sum* of (3.5.1) and (3.5.2). In [5] it is shown that for the linear s-sum,

$$\mu_0 = \mu_0^1 + \mu_0^2 - \sum_{p=1}^{s-1} \mu^1(v_p^1, v_{p+1}^1).$$

A subset $V = \{v_1, v_2, \ldots, v_s\}$ of nodes of the skeleton of an inequality is called *identifiable* if there exists as equality tour for the inequality that contains the family $2\{[v_i, v_{i+1}] \mid i \in \{1, 2, \ldots, s\}\}$. The following theorem is proven in [5].

Theorem 3.5.1. *Let (3.5.4) be obtained by linear s-sum of (3.5.1) and (3.5.2). If V^1 and V^2 are identifiable, the G is a skeleton of (3.5.4). The set W and all the subsets of V_{n_1} and V_{n_2} which are identifiable for (3.5.1) and (3.5.2), respectively, also are identifiable for (3.5.4).*

By Theorem 3.5.1 it is possible to show by induction that the graph obtained by repeatedly applying the s-sum operation is a skeleton for the corresponding inequality that is produced with this procedure.

The operation of composing facet defining inequalities for GTSP by linear s-sum preserves the properties required in (3.3).

Theorem 3.5.2. *Let (3.5.4) be obtained by linear s-sum of (3.5.1) and (3.5.2). If (3.3) holds for (3.5.1) and (3.5.2) with spanning trees S^1 and S^2, respectively, and if the subtree of S^1 induced by V^1 is isomorphic to the subtree of S^2 induced by V^2, then (3.3) holds also for (3.5.4).*

For simplicity let us call in this subsection any of the configurations of the sections 3.1, 3.2, and 3.3, a *path configuration* and let us call the corresponding inequality a *path inequality*. A path configuration (or a path inequality) is called *regular* if $n_i = n$, for $i = 1, 2, \ldots, k$. The nodes Y and Z are called the *odd* nodes of the configuration, all the other nodes are called *even*.

We consider now the 2-sum of path inequalities. In this case the only sets that qualify to be identifiable are the pairs $\{u_j^i, u_{j+1}^i\}$, for some $i \in \{1, 2, \ldots, k\}$ and $j \in \{0, \ldots, n_i\}$. When two nodes u' and u'' are identified in the 2-sum, the resulting node is even only if both u' and u'' are even, and odd in all the other cases.

A *regular parity path tree inequality* is defined recursively as one of the following two:

(i) a regular path inequality;
(ii) the 2-sum of a regular parity path tree inequality and a regular path inequality, with the condition that an odd node be identified only with an odd node.

A *regular parity path tree configuration* and a *directed regular parity path tree configuration* are defined analogously. In [5] it is shown that a regular parity path tree inequality is stable for the corresponding regular parity path tree configuration and that the coefficients of the crossing edges can be given in closed form. If (3.1) is a regular parity path tree inequality, the corresponding inequality (3.2) for the directed case is called a *regular parity s-path tree inequality*. Theorem 3.5.2 implies the following theorem.

Theorem 3.5.3. *Let D be any directed graph that contains a directed regular parity path tree configuration as a subgraph. Then the corresponding regular parity s-path tree inequality is facet defining for* GATSP(D).

3.6. Extensions of symmetric inequalities

Let

$$cx \geq c_0 \tag{3.6.1}$$

be a facet defining inequality for GTSP(K_n) and let e be an edge in E_n. We say that the inequality

$$c^*x^* \geq c_0^* \tag{3.6.2}$$

defined in $\mathbb{R}^{E_{n+2h}}$, with $h \geq 1$, is obtained from (3.6.1) on *cloning the edge* e (*h times*) in the following sense. The restriction of (3.6.2) to $\mathbb{R}^{E_n}$ coincides with (3.6.1). The remaining coefficients of (3.6.2) are defined as follows, where we assume without loss of generality that $e = [v_{n-1}, v_n]$ (see Figure 3.5 for $h = 1$):

$$\begin{aligned}
c_0^* &= c_0 + 2hc_e \\
c^*[v_i, v_{n+j}] &= c[v_i, v_{n-1}] \quad \text{for } 1 \leq i \leq n-2,\ 1 \leq j \leq 2h-1 \text{ and } j \text{ odd}, \\
c^*[v_i, v_{n+j}] &= c[v_i, v_n] \quad \text{for } 1 \leq i \leq n-2,\ 2 \leq j \leq 2h \text{ and } j \text{ even}, \\
c^*[v_{n+i}, v_{n+j}] &= 2c_e \quad \text{for } -1 \leq i < j \leq 2h \text{ and } j-i \text{ even}, \\
c^*[v_{n+i}, v_{n+j}] &= c_e \quad \text{for } -1 \leq i < j \leq 2h \text{ and } j-i \text{ odd}.
\end{aligned}$$

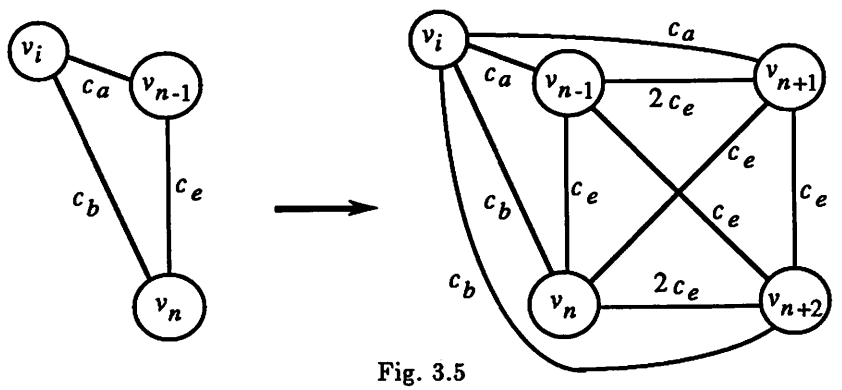


Fig. 3.5

An edge $e = [u, v] \in E_n$ is called *c-clonable* if the for every tour Θ of K_n, $c(\Theta) \geq c_0 + (t-2)c_e$, where t is the minimum of the degrees of u and v in Θ. We call a node $v \in V_n$, *q-critical* for (3.6.1) if for an optimal solution Θ^* of *gtsp* in $K_n - \{v\}$, when the weights are given by the restriction of (3.6.1), $c(\Theta^*) = c_0 - q$.

Theorem 3.6.1. *If $e = [v_{n-1}, v_n]$ is a c-clonable edge such that v_{n-1} and v_n are $2c_e$-critical for (3.6.1), then the following hold:*

(a) *(3.6.2) is facet defining for* GTSP(K_{n+2h});

(b) *if $f = [z_1, z_2] \neq e$ is an edge in E_n such that z_1 and z_2 are $2c_f$-critical for (3.6.1), then z_1 and z_2 are $2c_f^*$-critical for (3.6.2);*

(c) *if $G = (V_n, E)$ is a skeleton for (3.6.1), then the graph $G' = (V_{n+2h}, E')$ is a skeleton for (3.6.2), where*

$$\begin{aligned}
E' = E &\cup \left\{[v_{n+i}, v_{n+i+1}], [v_{n+i-2}, v_{n+i+1}], [v_{n+i-1}, v_{n+i}] \;\middle|\; i = 1, 3, \ldots, 2h-1\right\} \\
&\cup \left\{[w, u_{n+1}] \;\middle|\; [w, u_{n-1}] \in E\right\} \\
&\cup \left\{[w, u_{n+2}] \;\middle|\; [w, u_n] \in E\right\}.
\end{aligned}$$

The following theorem gives sufficient conditions under which cloning preserves the properties required in (3.3).

Theorem 3.6.2. *If (3.6.1) satisfies the conditions of Theorem 3.6.1, (3.3) holds for (3.6.1) and all the tours $\Theta(e_i)$ of (3.3) contain the edge e, then (3.3) holds also for (3.6.2).*

Let (3.1) be an inequality obtained by cloning some of the edges $e = [u_1^i, u_2^i]$, with $n_i = 2$, of a path (or a wheelbarrow, or a bicycle) inequality. We call (3.1) an *extended path* (or *wheelbarrow*, or *bicycle*) *inequality* and (3.2) an *extended s-path* (or *s-wheelbarrow*, or *s-bicycle*) *inequality.*

Theorem 3.6.3. *The extended s-path, s-wheelbarrow and s-bicycle inequalities are facet defining for* GATSP(D_n).

Let (3.1) be a an inequality obtained by cloning a *proper* subset of the diameters of a crown inequality. We call (3.1) an *extended crown inequality* and (3.2) an *extended s-crown inequality.*

Theorem 3.6.4. *The extended s-crown inequalities are facet defining for* GATSP(D_n).

4. Asymmetric facet defining inequalities for GATSP

In Section 3 we have shown several families of symmetric facet defining inequalities for GATSP(D). In this section we exhibit a family of asymmetric inequalities.

Consider the k-path configuration $P(n_1, n_2, \ldots, n_k) = (V_P, A_P)$ described earlier. Given $\overline{K} \subseteq \{1, 2, \ldots, k\}$ define the arc sets

$$YZ(\overline{K}) = \left\{(u_j^i, u_{j+1}^i) \mid i \in \overline{K};\ j \in \{0, \ldots, n_i\}\right\}$$
$$ZY(\overline{K}) = \left\{(u_{j+1}^i, u_j^i) \mid i \in \overline{K};\ j \in \{0, \ldots, n_i\}\right\}.$$

$YZ^m(\overline{K})$ (or $ZY^m(\overline{K})$) indicates that each arc in $YZ(\overline{K})$ ($ZY(\overline{K})$) has multiplicity m. For $i \in \{1, \ldots, k\}$ and $j^* \in \{0, \ldots, n_i\}$ define the arc set

$$A^i(j^*) = \left\{(u_j^i, u_{j+1}^i), (u_{j+1}^i, u_j^i) \mid j \neq j^* \in \{0, \ldots, n_i\}\right\}.$$

The extremal tours of $P(n_1, \ldots, n_k)$ can be classified into three classes.

(i) $\mathcal{ET}_1$: The extremal tours in this class have the following form

$$T = YZ(i^*) + ZY(i^*) + \left\{A^i(j_i) \mid i \neq i^* \in \{1, \ldots, k\};\ j_i \in \{0, \ldots, n_i\}\right\}.$$

(ii) $\mathcal{ET}_2$: The extremal tours in this class have the following form. For $K_1 \subseteq \{1, \ldots, k\}$, $K_2 \subseteq \{1, \ldots, k\}$, $|K_1| = |K_2| \geq 1$, $|K_1 \cap K_2| = 0$, $i \notin K_1 \cup K_2$, $j_i \in \{0, \ldots, n_i\}$ and $K_3 = \{1, \ldots, k\} - \{K_1 \cup K_2 \cup \{i\}\}$,

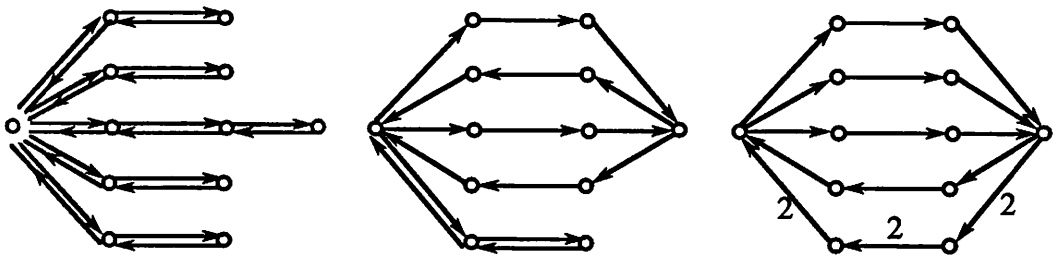
$$T = YZ(K_1) + ZY(K_2) + A^i(j_i) + \left\{A^\ell(j_\ell) \mid \ell \in K_3;\ j_\ell \in \{0, \ldots, n_\ell\}\right\}.$$

(iii) $\mathcal{ET}_3$: The extremal tours in this class have the following form. For $K_1 \subseteq \{1, \ldots, k\}$, $|K_1| = r < \lfloor \frac{k}{2} \rfloor$; $K_2 \subseteq \{1, \ldots, k\}$, $|K_2| = r + 2s$, for $s \geq 1$; $|K_1 \cap K_2| = 0$, $K_1 \cup K_2 \subseteq \{1, \ldots, k\}$, $i \notin K_1 \cup K_2$ and $K_3 = \{1, \ldots, k\} - \{K_1 \cup K_2 \cup \{i\}\}$,

$$T = YZ(K_1) + ZY(K_2) + ZY^{2s}(i) + \left\{A^\ell(j_\ell) \mid \ell \in K_3;\ j_\ell \in \{0, \ldots, n_\ell\}\right\}$$

or

$$T = YZ(K_2) + ZY(K_1) + YZ^{2s}(i) + \left\{A^\ell(j_\ell) \mid \ell \in K_3;\ j_\ell \in \{0, \ldots, n_\ell\}\right\}.$$



(i) (ii) (iii)

Fig. 4.1.

For $k = 5$, examples of tours from each class are as shown in Figure 4.1. For the example shown we have taken $|K_3| = 0$ and $s = 1$.

Given any directed graph $D = (V, A)$ one can construct an undirected graph $G = (V, E)$ where $e = [u_i, u_j] \in E$ if and only if $|\{(u_i, u_j), (u_j, u_i)\} \cap A| \geq 1$. Given any tour T on D one can construct a tour Θ on G where the multiplicity of $e = [u_i, u_j]$ in Θ is the sum of the multiplicity of (u_i, u_j) and (u_j, u_i) in T. Using this procedure the extremal tours from $\mathcal{ET}_1$ and $\mathcal{ET}_2$ give tours that are also extremal in the undirected case (see [2]). However the extremal tours from $\mathcal{ET}_3$ have no counterpart in the undirected case since the corresponding undirected tour is not extremal.

Let T be a tour of $D = (V, A)$. For an ordered triple $\langle u, v, w \rangle$ of distinct nodes of V, we call *shortcut* for T the vector $s_{u,v,w} \in \mathbb{R}^A$, defined by

$$s_{u,v,w}(a) = \begin{cases} 1 & \text{if } a = (u, w), \\ -1 & \text{if } a = (u, v) \text{ or } a = (v, w), \\ 0 & \text{otherwise,} \end{cases}$$

such that the family T' having incidence vector $\chi^{T'} = \chi^T + s_{u,v,w}$ is either a tour of D or a tour of $D - \{v\}$ and v is isolated in $D[T']$. T' is said to be obtained on *reducing* T *by the shortcut* $s_{u,v,w}$.

All the extremal tours of $W(n_1, n_2, \ldots, n_k)$ can be obtained from those of $P(n_1, n_2, \ldots, n_k)$ by repeatedly applying a reduction with shortcut $s_{u,Z,u'}$, where the set $\{u, u'\}$ coincides with $\{u^i_{n_i}, u^{i+1}_{n_{i+1}}\}$, for $i \in \{1, \ldots, k\}$. All the extremal tours of $B(n_1, n_2, \ldots, n_k)$ can be obtained from those of $P(n_1, n_2, \ldots, n_k)$ by repeatedly applying a reduction with shortcuts $s_{u,Z,u'}$ and $s_{v,Y,v'}$, where the set $\{u, u'\}$ coincides either with $\{u^i_{n_i}, u^{i+1}_{n_{i+1}}\}$ or with $\{u^i_{n_i}, u^{i+2}_{n_{i+2}}\}$, for $i \in \{1, \ldots, k\}$, and the set $\{v, v'\}$ coincides either with $\{u^i_1, u^{i+1}_{n_1}\}$ or with $\{u^i_1, u^{i+2}_{n_1}\}$, for $i \in \{1, \ldots, k\}$. Here as usual all the superscript indices are taken modulo k, where $0 = k$.

4.1. The a-path inequalities

The s-path inequality (3.1.1) is satisfied at equality only by tours from $\mathcal{ET}_2$. The asymmetric inequality introduced here is also satisfied at equality by some tours from the class $\mathcal{ET}_3$. Let $D = (V_P, A)$ be any graph that contains the k-path configuration $P(n_1, \ldots, n_k)$ as a subgraph. An *a-path inequality* associated with $P(n_1, \ldots, n_k)$ is the inequality

$$cx \geq c_0 \tag{4.1.1}$$

on $\mathbb{R}^A$, where c and c_0 are defined as follows.

$$c_0 = 2 + \sum_{i=2}^{k} \frac{n_i + 1}{n_i - 1},$$

$$c(Y, Z) = 1 + \frac{1}{n_1},$$

$$c(Z, Y) = 1 - \frac{1}{n_1},$$

$$c(Y, u_j^1) = c(u_j^1, Y) = \frac{j}{n_1} \quad \text{for all } j \in \{1, \ldots, n_1\},$$

$$c(Y, u_j^i) = \frac{j}{n_i - 1} + \frac{1}{n_1} \quad \text{for all } i \in \{2, \ldots, k\};\ j \in \{1, \ldots, n_i\},$$

$$c(u_j^i, Y) = \frac{j}{n_i - 1} - \frac{1}{n_1} \quad \text{for all } i \in \{2, \ldots, k\};\ j \in \{1, \ldots, n_i\},$$

$$c(u_j^1, u_q^1) = \frac{|j - q|}{n_1} \quad \text{for all } j \neq q \in \{0, \ldots, n_i + 1\},$$

$$c(u_j^i, u_q^i) = \frac{|j - q|}{n_i - 1} \quad \text{for all } i \in \{2, \ldots, k\};\ j \neq q \in \{1, \ldots, n_i + 1\},$$

$$c(u_j^i, u_q^r) = \frac{1}{n_i - 1} + \frac{1}{n_r - 1} + \left| \frac{j - 1}{n_i - 1} - \frac{q - 1}{n_r - 1} \right|$$
$$\text{for all } i \neq r \in \{2, \ldots, k\};\ j \in \{1, \ldots, n_i\};\ q \in \{1, \ldots, n_r\},$$

$$c(u_j^1, u_q^r) = \frac{1}{n_r - 1} + \frac{1}{n_1} + \left| \frac{j}{n_1} - \frac{q - 1}{n_r - 1} \right|$$
$$\text{for all } r \in \{2, \ldots, k\};\ j \in \{1, \ldots, n_1\};\ q \in \{1, \ldots, n_r\},$$

$$c(u_j^i, u_q^1) = \frac{1}{n_i - 1} + \left| \frac{j - 1}{n_i - 1} - \frac{q - 1}{n_1} \right|$$
$$\text{for all } i \in \{2, \ldots, k\};\ j \in \{1, \ldots, n_i\};\ q \in \{1, \ldots, n_1\}.$$

In the way the coefficients are defined the bidirected path $YZ(1) + ZY(1)$ is different from the other bidirected paths. In general one can replace 1 by any $i \in \{2, \ldots, k\}$. This only involves a renumbering of the paths with the coefficients remaining unchanged.

Theorem 4.1.1. *The a-path inequality (4.1.1) is facet defining for* GATSP(D).

4.2. The a-wheelbarrow inequalities

The s-wheelbarrow inequality is the restriction of the s-path inequality to arcs in the wheelbarrow configurations. In this subsection we define the a-wheelbarrow inequality that is a similar restriction of the a-path inequality.

Consider any directed graph $D = (V_W, A)$ that contains the directed k-wheelbarrow configuration $W(n_1, n_2, \ldots, n_k)$ as a subgraph. An ***a-wheelbarrow inequality*** associated with $W(n_1, n_2, \ldots, n_k)$ is the inequality

$$cx \geq c_0 \tag{4.2.1}$$

on $\mathbb{R}^A$, where c and c_0 are as defined for the a-path inequality (4.1.1). The inequality (4.2.1) is a restriction of (4.1.1).

Theorem 4.2.1. *The a-wheelbarrow inequality (4.2.1) is facet defining for* GATSP(D).

4.3. The a-bicycle inequalities

In this subsection we proceed as for the a-wheelbarrow inequalities.

Consider any directed graph $D = (V_B, A)$ that contains the directed k-bicycle configuration $B(n_1, n_2, \ldots, n_k)$ as a subgraph. An ***a-bicycle inequality*** associated with $B(n_1, n_2, \ldots, n_k)$ is the inequality

$$cx \geq c_0 \tag{4.3.1}$$

on $\mathbb{R}^A$, where c and c_0 are as defined for the a-path inequality (4.1.1). The inequality (4.3.1) is a restriction of (4.1.1).

Theorem 4.3.1. *The a-bicycle inequality (4.3.1) is facet defining for* GATSP(D).

Conclusions

We have defined several inequalities and shown that they define facets of GATSP. Some of these inequalities have asymmetric coefficients and hence they do not have a counterpart in GTSP. Other inequalities have symmetric coefficients and can be derived from facet defining inequalities for GTSP. These facet defining inequalities for GTSP are the *path* inequalities [2] and their extensions [7] (the classical *comb* (see, e.g., [4]) and *chain* [9] inequalities are a small subset of these inequalities), the *path-tree* inequalities [5] (the classical *clique-tree* inequalities (see, e.g., [4]) are a small subset of these inequalities), and the *crown* inequalities [8].

In a sequel paper we show how most of the inequalities described here define facet of ATSP as well. To do so we exploit the fact that GATSP is a relaxation of ATSP and we use a proof technique similar to the one used in [6] and [7] for the undirected case.

References

[1] S. Chopra and G. Rinaldi, "The Graphical Asymmetric Traveling Salesman Polyhedron" Preprint IASI-CNR, (Rome, March 1990).

[2] G. Cornuéjols, J. Fonlupt and D. Naddef, "The traveling salesman problem on a graph and some related integer polyhedra", *Mathematical Programming* 33 (1985) 1–27.

[3] B. Fleischmann, "A new class of cutting planes for the symmetric travelling salesman problem", *Mathematical Programming* 40 (1988) 225–246.

[4] M. Grötschel and M. Padberg, "Polyhedral theory", in E.L. Lawler et al., eds., The Traveling Salesman Problem (Wiley & Sons, Chichester, 1985) pp. 251–305.

[5] D. Naddef and G. Rinaldi, "The symmetric traveling salesman polytope and its graphical relaxation: Composition of valid inequalities", Report RR 719-M, ARTEMIS-IMAG, Université Joseph Fourier (Grenoble, May 1988), to appear in *Mathematical Programming.*

[6] D. Naddef and G. Rinaldi, "The graphical relaxation: a new framework for the symmetric traveling salesman polytope", Report R.244, IASI-CNR (Rome, 1988).

[7] D. Naddef and G. Rinaldi, "The symmetric traveling salesman polytope: New facets from the graphical relaxation", Report R.248, IASI-CNR (Rome, 1988).

[8] D. Naddef and G. Rinaldi, "The crown inequalities for the symmetric traveling salesman polytope", Report R.249, IASI-CNR (Rome, December 1988), to appear in *Mathematics of Operations Research.*

[9] M. Padberg and S. Hong, "On the symmetric travelling salesman problem: A computational study", *Mathematical Programming Study* 12 (1980) 78–107.

A DECOMPOSITION THEOREM FOR BALANCED MATRICES

Michele Conforti*
Gérard Cornuéjols**

ABSTRACT A 0,1 matrix is *balanced* if it contains no square submatrix of odd order with two ones per row and per column. This concept was introduced by Berge and plays an important role in combinatorial optimization. Consider a balanced matrix with at least three rows and three columns. In a forthcoming paper, we show that the associated bipartite incidence graph has a *double star cutset* , i.e. the graph can be disconnected by removing two nodes and a subset of their neighbors. In this paper we outline the proof of this result and we provide a complete proof in the case where the bipartite incidence graph contains a wheel.

* Dipartimento di Matematica, Università di Trento, 38050 Povo (Trento), Italy

** Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pa 15213.

This work was supported in part by NSF grants DDM-8800281 and ECS-8901495. It was made possible by the hospitality of the Mathematics Department, University of Trento and of the Graduate School of Industrial Administration, Carnegie Mellon University, who made this cooperation occur under ideal conditions.

1 INTRODUCTION

A 0,1 matrix is *balanced* if it contains no square submatrix of odd order with two ones per row and per column. This concept was introduced by Berge [1] and is of great interest in combinatorial optimization since balanced matrices are exactly the 0,1 matrices A such that the associated set covering polyhedron

$$\{x \geq 0 : Ax \geq b\}$$

has integral extreme points for every 0,1 vector b, see Berge [1] and Fulkerson, Hoffman and Oppenheim [10]. Little is known about the structural properties of balanced matrices, other than the definition. The goal of this paper is to describe such properties. The results are in the spirit of those that Conforti and Rao [8] proved for linear balanced matrices. (A 0,1 matrix is *linear* if it does not contain a 2x2 submatrix of all ones).

Given a 0,1 matrix A, let $G_A=(V^- \cup V^+,E)$ be the bipartite graph having a node in V^- for every row of A, a node in V^+ for every column of A and an edge joining nodes $i \in V^-$ and $j \in V^+$ if and only if the entry a_{ij} of A equals 1.

Conversely, let $(V^- \cup V^+, E)$ be a bipartite graph. We assume in this paper that $V^- \neq \emptyset$, $V^+ \neq \emptyset$, and E contains no parallel edges. Then, there is a unique (up to a permutation of rows and columns) 0,1 matrix A such that $G_A=(V^- \cup V^+, E)$. Therefore it is equivalent to study 0,1 matrices and bipartite graphs. We prefer to work with bipartite graphs since we make extensive use of graphical notions such as paths and cycles. The bipartite graph G_A is said to be *balanced* if and only if the 0,1 matrix A is balanced.

Here, we assume that all paths and cycles are elementary, i.e. they do not contain repeated nodes. The *length* of a path (cycle) is its number of edges. A *chord* of a path (cycle) is an edge connecting two nonconsecutive nodes of the path (cycle). A chordless cycle in a graph is called a *hole*. It follows from the definition that a bipartite graph is balanced if and only if contains no hole of length 2 modulo 4.

The node set of a subgraph X of G is denoted by V(X). Certain important graphs are called *configurations*. When we say G *contains the configuration* X, we mean that X is a node induced subgraph of G. Given $S \subseteq V(G)$, we denote by $G \backslash S$ the graph induced by the nodes in $V(G) \backslash S$. Node u is a *neighbor* of node v if u and v are joined by an edge. We say that a node $v \in V(G) \backslash V(X)$ is *strongly adjacent* to a configuration X contained in G if v has at least two neighbors in V(X). N(v) denotes $\{v\}$ union the set of all the neighbors of v. Let $S \subseteq V(G)$ be a node set such that the graph $G \backslash S$ is disconnected. We say that S is a *star cutset* if $S \subseteq N(v)$ for some $v \in S$. We say that S is a *double star cutset* if $S \subseteq N(a) \cup N(b)$ for $a,b \in S$.

If $G=(V^- \cup V^+, E)$ satisfies $|V^-| \leq 2$ or $|V^+| \leq 2$, then it follows from the definition that G is balanced. The main result of this paper states that all other balanced graphs have a double star cutset.

Theorem 1.1 If $G=(V^- \cup V^+, E)$ is balanced and $|V^-|$, $|V^+| \geq 3$, then G has a double star cutset.

Note that, although double star cutsets can be identified in polynomial time, this theorem does not yield a polynomial algorithm for checking balancedness (at least not directly), for at least two reasons. In decomposing G, the nodes of the double star cutset must appear in each connected component of the decomposed graph. Since the decomposition step is repeated, it is not clear that a polynomial number of pieces will result. The other problem is that the converse of Theorem 1.1 is not true, in other words, it is possible that all pieces are balanced but G is not balanced. In a recent work, Conforti, Cornuéjols and Rao [6] have overcome these difficulties, by exploiting a strengthening of Theorem 1.1.

In a bipartite graph, every cycle has length 2k, for some integer $k \geq 2$. When k is even, we say that the cycle is *par*; when k is odd, we say that it is *unpar*. If $G_A=(V^- \cup V^+, E)$ does not contain any unpar cycle, then the corresponding 0,1 matrix A is known as *restricted unimodular* and Theorem 1.1 holds (Yannakakis [11]). See Commoner [4], Conforti and Conuéjols [5] and Conforti and Rao [7] for further results on restricted unimodular

matrices. Consequently, to prove Theorem 1.1, we need to consider bipartite graphs that contain an unpar cycle but no unpar hole.

We say that an unpar cycle is *minimal* if no subset of its nodes induces an unpar cycle. Conforti and Rao [8] have shown that a minimal unpar cycle can have at most three chords. Furthermore, if it has three chords it must be of length 6, and if it has two chords, say (s,t), (u,v) with $s,u \in V^-$ and $t,v \in V^+$, then s, v, t, u appear in that order in the cycle and (s,v) and (t,u) are edges: The two chords are said to be *crossing and neighborly*, see Figure 1.1. In this figure and in all the subsequent ones, dotted lines reperesent paths of length at least 1, solid lines represent edges.

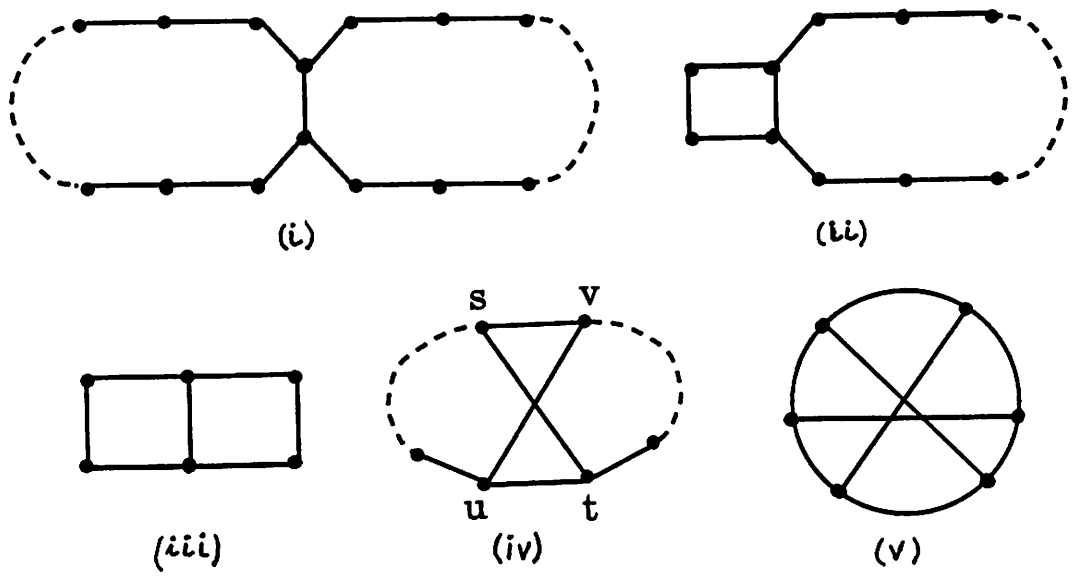


Figure 1.1 Minimal unpar cycles with at least one chord.

If $G=(V^- \cup V^+,E)$ only contains minimal unpar cycles of Types (iv) and (v), then the corresponding 0,1 matrix A is said to be *strongly unimodular* and Theorem 1.1 is true (Conforti and Rao [7]). If G only contains minimal unpar cycles of Types (iii), (iv) and (v), then Theorem 1.1 has been shown by Carducci [3]. To prove Theorem 1.1 in general, we introduce the concepts of an even wheel and a parachute, see Figure 1.2.

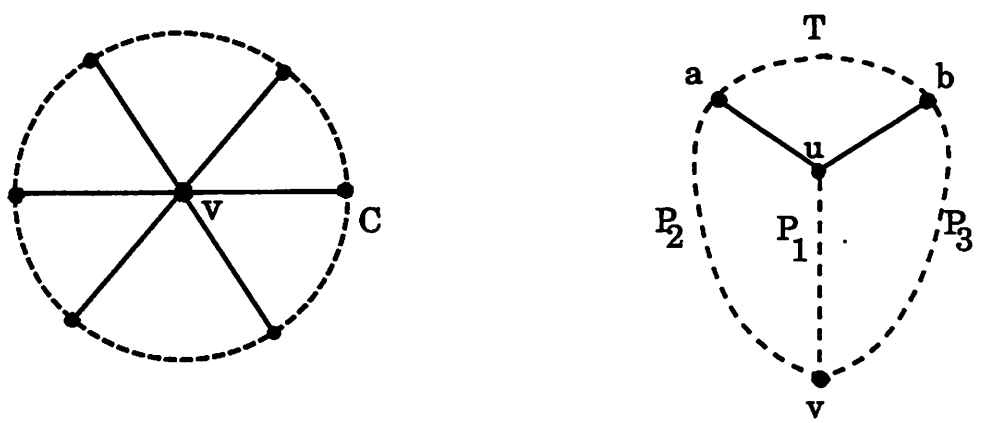


Figure 1.2 Even wheel and parachute

An *even wheel* (C,v) is defined by a node v and a par hole C not containing v but containing an even number, greater than or equal to 4, of neighbors of v.
A *parachute* (P_1,P_2,P_3,T) is defined by four chordless paths P_1=(v,...,u), P_2=(v,...,a), P_3=(v,...,b), T=(a,...,b), each of length at least 2, and with distinct intermediate nodes, such that a and b are adjacent to u. Furthermore, the graph induced by $V(P_1)\cup V(P_2)\cup V(P_3)\cup V(T)$ only contains the edges stated above, namely (a,u), (b,u) and the edges of P_1, P_2, P_3 and T.

Our proof of Theorem 1.1 is obtained through the following sequence of results.

Theorem 1.2 If $G=(V^-\cup V^+,E)$ is balanced, $|V^-|$, $|V^+|\geq 3$, and G contains no minimal unpar cycle of type (ii), then G has a double star cutset.

The next step of the proof is to show the following result.

Theorem 1.3 If G is a balanced bipartite graph containing an unpar cycle of type (ii), then either
(a) G has a double star cutset, or
(b) G contains an even wheel, or
(c) G contains a parachute.

We then show that, if G contains an even wheel or a parachute, it has a double star cutset. The proof is obtained by the following sequence of results.

Theorem 1.4 If G is a balanced bipartite graph containing an even wheel, then G has a double star cutset.

For the next theorem, we need to define two more configurations, see Figure 1.3.

Connected squares (P_1,P_2,P_3,P_4) are defined by four chordless paths $P_i=(u_i,.....,v_i)$, $1\leq i\leq 4$, each of length at least 2, with distinct intermediate nodes, such that (u_1,u_2,u_3,u_4) and (v_1,v_2,v_3,v_4) each define a 4-cycle (*square*). Furthermore, the graph induced by $V(P_1)\cup V(P_2)\cup V(P_3)\cup V(P_4)$ contains no other edge that those described above, namely the edges of the paths P_i, $1\leq i\leq 4$, and (u_1,u_2), (u_2,u_3), (u_3,u_4), (u_4,u_1), (v_1,v_2), (v_2,v_3), (v_3,v_4), (v_4,v_1).

Goggles (C,T) are defined by an unpar odd cycle C of type (iv), say with neighborly crossing chords (u,a) and (x,b) and a chordless path T=(h,...,v) where the intermediate nodes of T do not belong to C and, in addition, the nodes h, v belong to different components of C\{u, x, a, b}. Furthermore, the nodes u, x, a, b, are not adjacent to h or v. The path T may be reduced to a single edge. Finally, the only edges of the graph induced by $V(C)\cup V(T)$ are those of C, T and (u,a), (x,b).

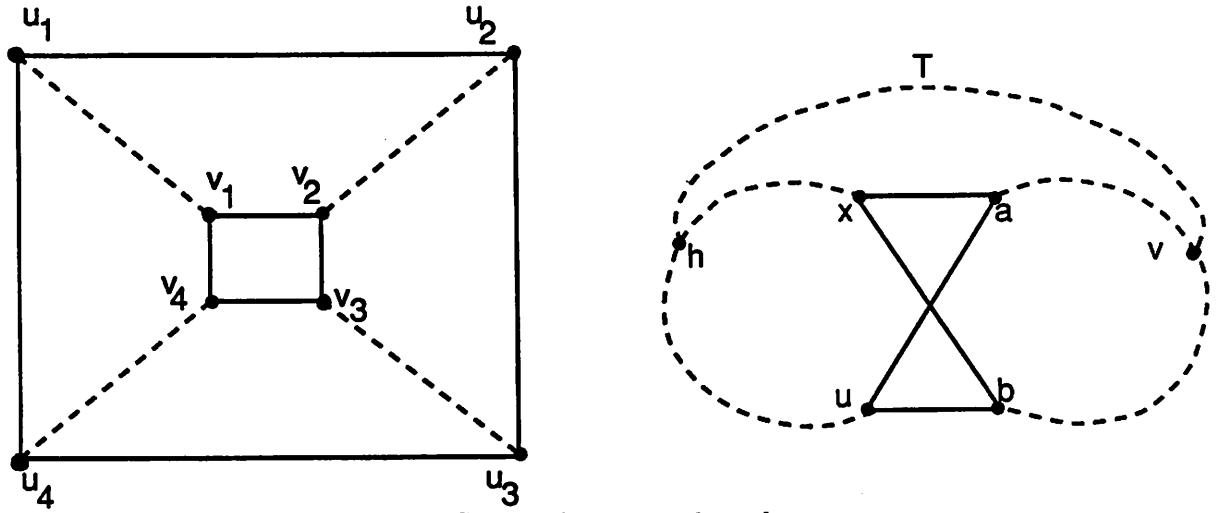


Figure 1.3 Connected squares and goggles

Theorem 1.5 If G is a balanced bipartite graph containing a parachute, then either

(a) G has a double star cutset, or

(b) G contains connected squares, or

(c) G contains goggles.

Theorem 1.6 If G is a balanced bipartite graph containing connected squares, then G has a double star cutset .

Theorem 1.7 If G is a balanced bipartite graph containing goggles, then G has a double star cutset.

These theorems imply Theorem 1.1. Our proofs of Theorems 1.2, 1.3, 1.5, 1.6 and 1.7 are quite long and will be given in a forthcoming paper. In this paper, we prove Theorem 1.4. The proof is divided into two parts, which are given in Sections 2 and 3 respectively.

To prove Theorems 1.2-1.7, we need a clear idea of the various types of strongly adjacent nodes that can occur for each of the configurations given in Figures 1.1-1.3 above. In the case of unpar cycles of Type (i)-(iii), the strongly adjacent nodes that can occur have been described by Conforti and Rao [8], see Figure 1.4.

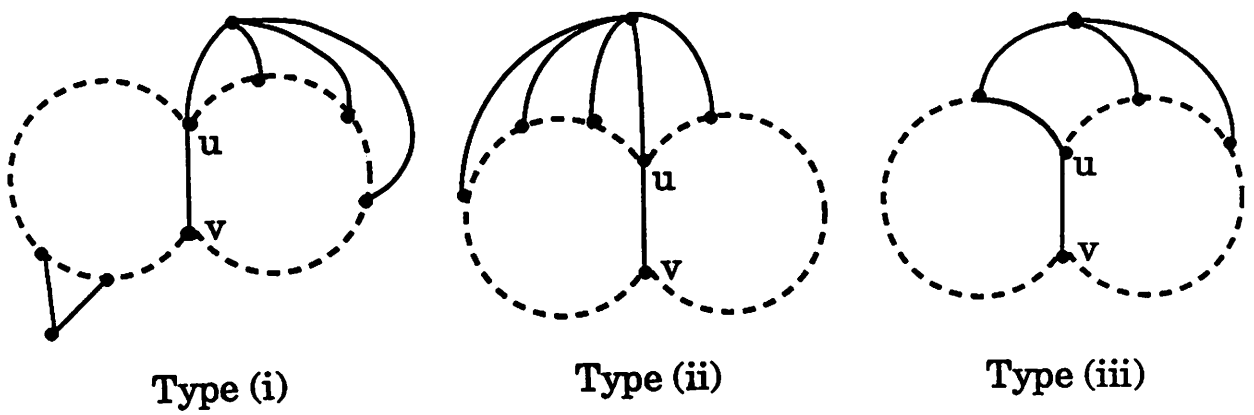


Figure 1.4 Different types of strongly adjacent nodes to an unpar cycle with a unique chord.

Let C be an unpar cycle with a unique chord (u,v). The strongly adjacent nodes to C can be divided into 3 types. The Type (i) strongly adjacent nodes have an even number of neighbors in C but no neighbor in one of the two components of C\{u,v}. Type (ii) strongly adjacent nodes have either u or v as neighbor and an odd number of neighbors in each component of C\{u,v}. Finally, the Type (iii) strongly adjacent nodes have an even number of neighbors in one component of C\{u,v} and, in the other component, they have only one neighbor which is adjacent to either u or v.

Two configurations always contain unpar holes and will be very useful in the proofs: the 3-path configuration and the odd wheel configuration, see Figure 1.5. A *3-path configuration* is defined by three chordless paths P_1, P_2, P_3, each of length at least 3, joining nodes u and v in opposite sides of the bipartition. Except for u and v, the three paths are node-disjoint and the graph induced by the nodes of P_1, P_2, P_3 contains no other edge than those of the paths. An *odd wheel* (C,w) is defined by a hole C and a node w not belonging to C but adjacent to an odd number, greater than one, of nodes of C.

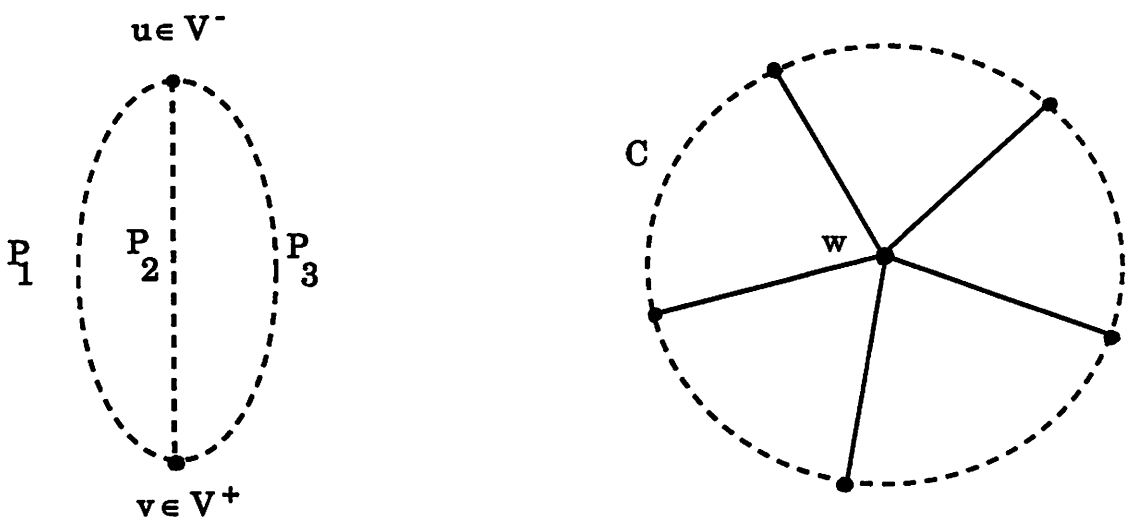


Figure 1.5 3-path configuration and odd wheel

It is easy to check that the configurations of Figure 1.5 contain unpar holes: Assume the 3-path configuration does not. Then P_1 must have length 1 (mod 4) and P_2 length 3 (mod 4) or vice-versa. Then P_3 closes an unpar hole with one of the first two paths. Assume the odd wheel configuration (C,v) does not contain an unpar hole. Any sector of C between two consecutive neighbors of v must have length 2 (mod 4), else it would close an unpar hole with v. This implies that C is an unpar hole, since it is composed by an odd number of such sectors.

The spirit of our proof is that, if a bipartite graph contains a given configuration (an even wheel, goggles and so forth) but no double star cutset, then it also contains a 6-hole or one of the two configurations of Figure 1.5 and therefore it is not balanced.

To emphasize the fact that the graphs of Figure 1.5 are central to the study of balanced matrices, we mention a beautiful result of Truemper. A 0,+1,-1 matrix is said to be balanced if in every square submatrix with two nonzero entries per row and per column the sum of the entries is 0 (mod4). To *sign* a 0,1 matrix means to replace some of the 1's by -1's.

Theorem 1.8 (Truemper [11]) A 0,1 matrix A can be signed into a balanced 0,+1,-1 matrix if and only if the bipartite graph G_A contains neither a 3-path configuration nor an odd wheel configuration.

2. STRONGLY ADJACENT NODES TO AN EVEN WHEEL

Let (W,v) be an even wheel of a balanced bipartite graph G. Throughout this section and in the next one, we assume that $v \in V^+$. We denote by W^v the set of nodes of W adjacent to v . A subpath of W having two nodes of W^v as endpoints and only nodes of $V(W) \setminus W^v$ as intermediate nodes is called a *sector* of (W,v). Two sectors are *adjacent* if they have a common endpoint and two nodes of W^v are *consecutive* if they are the endpoints of some sector. In our exposition, we paint the nodes of $V(W) \setminus W^v$ with two colors, say blue and green in such a way that, if two nodes of $V(W) \setminus W^v$ are in the same sector, they have the same color and if they are in adjacent sectors, they have distinct colors. The nodes of W^v are left unpainted.

The goal of this section is to prove the following two theorems about the strongly adjacent nodes to W.

Theorem 2.1 Let $u \in V \setminus N(v)$ be a node with neighbors in at least two distinct sectors of W. Then u has exactly two neighbors in W, say u_j and u_k. Furthermore, the nodes u_j, u_k belong to sectors of the same color and at least one of u_j, u_k is not adjacent to unpainted nodes.

Define the set of nodes:

$A(W,v)=\{u \in V^+ \mid$ No sector of (W,v) entirely contains the set W^u and $|W^v \cap W^u| \geq 2\}$.

Theorem 2.2 If a balanced bipartite graph contains an even wheel, then it contains an even wheel (W,v) such that

$$\left| \bigcap_{u \in A(W,v)} W^u \right| \geq 2.$$

Proof of Theorem 2.1: Assume u has neighbors in at least three different sectors, say S_i, S_j, S_k. If none of these sectors is adjacent to the other two, then there exist three unpainted nodes v_i, v_j, v_k, such that $v_i \in V(S_i) \setminus (V(S_j) \cup V(S_k))$, $v_j \in V(S_j) \setminus (V(S_i) \cup V(S_k))$, $v_k \in V(S_k) \setminus (V(S_i) \cup V(S_j))$. This implies the existence of a 3-path configuration from u to v, where each of the nodes v_i, v_j, v_k belongs to a distinct such path, see Figure 2.1.

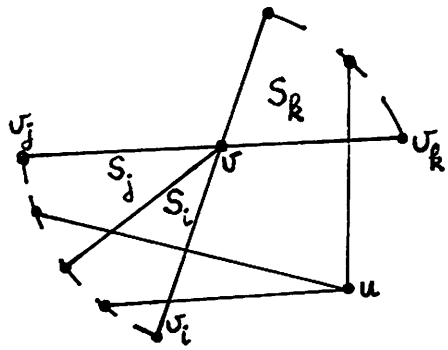


Figure 2.1

Now assume some sector is adjacent to the other two, say S_j is adjacent to both S_i and S_k. Then, with the notation of Figure 2.2, there is a 3-path configuration from u to v unless node u has a neighbor u_i in S_i adjacent to v_i and a neighbor u_k in S_k adjacent to v_k. When this is the case, the nodes u, u_i, v_i, v, v_k, u_k induce a 6-hole.

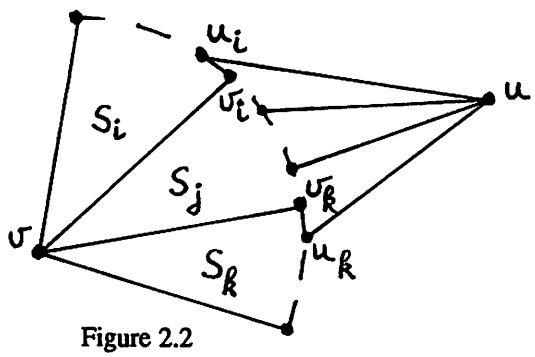


Figure 2.2

So u has neighbors in at most two different sectors of the wheel, say S_j and S_k. If these two sectors are adjacent, then denote by v_j and v_k the endpoints such that $v_j \in V(S_j) \setminus V(S_k)$ and $v_k \in V(S_k) \setminus V(S_j)$ respectively. Among the nodes of $N(u) \cap V(S_j)$, let u_j be the one such that the subpath of S_j connecting u_j to v_j is shortest. Similarly $u_k \in N(u) \cap V(S_k)$ has the shortest subpath to v_k in S_k. Let P' be the subpath of W joining u_j to u_k and containing the common endpoint of S_j and S_k. Now, consider the hole W' obtained from W by replacing P' by the path (u_j,u,u_k). The wheel (W',v) is an odd wheel.

So the sectors S_j and S_k are not adjacent. If u has three neighbors or more on W, say two or more in S_j and at least one in S_k, then denote by v_j and v_{j-1} the endpoints of S_j and by v_k one of the endpoints of S_k. There exists a 3-path configuration between u and v where each of the nodes v_j, v_{j-1} and v_k belongs to a different path.

Therefore u has only two neighbors in W, say $u_j \in V(S_j)$ and $u_k \in V(S_k)$. Let C_1 and C_2 be the holes formed by the node u and the two subpaths of W connecting u_j to u_k, respectively. In order for both (C_1,v) and (C_2,v) to be even wheels, the sectors S_j and S_k must be of the same color.

Finally, assume that u_j is adjacent to an endpoint of S_j, say v_j, and that u_k is adjacent to an endpoint of S_k, say v_k. Then u, v, u_j, v_j, u_k, v_k induce a 6-hole. ♦

Before proving Theorem 2.2, we need the following results about the structure of the nodes of V^+ which are strongly adjacent to W. We first introduce a classification of such nodes $u \in V^+$, relative to a chosen center v of the wheel.

Type 1: There exists a sector of (W,v) containing all the nodes of W^u.
Type 2: Node u is not of Type 1 and all its neighbors in W are unpainted. Note that, in particular, the center v of the wheel is of Type 2.
Type 3: Node u is not of Types 1 or 2 and all its painted neighbors in W have the same color.
Type 4: Node u has painted neighbors of both colors.

We first study the structure of the Type 4 nodes.

Lemma 2.3 Let u be a Type 4 strongly adjacent node to an even wheel (W,v). Let s and t be a green and a blue neighbor of u, respectively. Each of the subpaths of W joining s to t contains at least one unpainted neighbor of u. Hence $u \in A(W,v)$.

Proof: Assume that one of the two subpaths of W joining s to t contains no unpainted neighbor of u. Let Q be this subpath. Let P be a subpath of Q starting at a green neighbor of u, say s', ending at a blue neighbor of u, say t', and containing no other painted neighbor of u. P contains an odd number of unpainted nodes, none of which are adjacent to u. If this number is three or more, then v is the center of an odd wheel with hole induced by the nodes of P and u.

So P contains exactly one neighbor of v, say x. Consider the unpar cycle C with unique chord (v,x) induced by v and the two sectors having x as an endpoint. The node u is a strongly adjacent node relative to C and therefore must be of one of the three types shown in Figure 1.4. It is not of Type (i) since s' and t' are in different sectors. It is not of Type (ii) either since u is not adjacent to u or x. Since s' and t' are painted, they are not adjacent to v and since $s',t',x \in V^-$, the nodes s' and t' are not adjacent to x either. So the node u is not of Type (iii) relative to C. This contradicts the balancedness assumption. Therefore Q must contain an unpainted node adjacent to u. ♦

Lemma 2.4 If a node of V^+, strongly adjacent to W, has a unique neighbor in a sector, then this node is unpainted.

Proof: Assume that node u has a unique neighbor u_i in sector S_i and that u_i is painted. Let v_i and v_{i-1} be the endpoints of S_i. Since u is strongly adjacent to W, then u has at least one neighbor in the path $W \backslash V(S_i)$. Choose u^* among the nodes of $W^u \backslash V(S_i)$ and choose v^* among the nodes of $W^v \backslash V(S_i)$ in such a way that the subpath joining

u^* to v^* in $W \setminus V(S_i)$ is shortest. Note that $u^* \in V^-$, hence u^* cannot be adjacent to v_i or v_{i-1}. This implies that $u_i \in V^-$ is connected to $v \in V^+$ by a 3-path configuration, where each of the nodes v_i, v_{i-1} and v^* belongs to a different path. ♦

We now consider a wheel (W,v) of G with the following property.

Property 2.5 There exist no nodes x, y, z of Types 1, 2, 3 or 4 such that the graph induced by $V(W) \cup \{x,y,z\}$ contains a wheel (W',u) where W' is a shorter hole than W and u is one of the nodes x, y or z.

Note that Property 2.5 does not depend on v. It is a property of the par hole W. For example, the shortest par hole W for which there exists a node $v \in V(G) \setminus V(W)$ with more than two neighbors in W, satisfies Property 2.5. The following remarks are an immediate consequence of Property 2.5.

Remark 2.6 Every strongly adjacent node to W which does not belong to N(v) has at most two neighbors in each sector of (W,v).

Remark 2.7 Every Type 1 node u has exactly two neighbors in W, say u' and u", and there is a painted node in W which is adjacent to both u' and u".

Remark 2.8 For every Type 2 node u, $W^u = W^v$.

Remark 2.9 Every Type 3 node u has exactly two neighbors either in each green sector or in each blue sector. Hence, $|W^u| = |W^v|$.

Lemma 2.10 Let (W,v) be an even wheel satisfying Property 2.5. Every Type 4 node u satisfies $|W^u| = |W^v|$.

Proof: By Lemma 2.3, we have that $|W^u \cap W^v| \geq 2$. Let P be a path in W with endpoints in $W^u \cap W^v$, say x and y, but no intermediate node in $W^u \cap W^v$. The nodes x and y are said to be consecutive nodes of $W^u \cap W^v$ in W. Now Lemma 2.3 implies that $V(P) \cap N(u)$ does not contain nodes of distinct colors. Assume w.l.o.g. that $V(P) \cap N(u)$ contains no green node. Then Property 2.5 implies that u has exacly two neighbors in every blue sector of P. This shows that $|W^u \cap V(P)| = |W^v \cap V(P)|$. By repeating the argument between any pair of consecutive nodes of $W^u \cap W^v$ in W, we get the equality claimed in the lemma. ♦

Lemma 2.11 Let (W,v) be an even wheel satisfying Property 2.5 and let u be a Type 4 node having painted neighbors in two adjacent sectors, say S_i, S_{i+1}. Then every Type 2, 3 or 4 node is adjacent to the common endpoint of S_i, S_{i+1}.

Proof: We use the notation of Figure 2.3. Node v_i belongs to N(u), as a consequence of Lemma 2.3. Assume by contradiction that there exists a node w, of Type 2, 3 or 4, which is not adjacent to v_i.

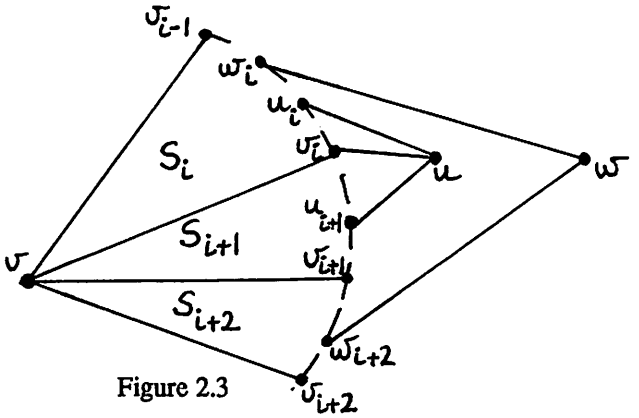


Figure 2.3

By Property 2.5, node w has a painted neighbor in S_i or S_{i+1}. If w has a painted neighbor in both S_i and S_{i+1}, then Lemma 2.3 implies that w is adjacent to v_i. Therefore we assume w.l.o.g. that w has a painted neighbor in S_i but no painted neighbor in S_{i+1}. Property 2.5 implies that w has a neighbor in S_{i+2}. Let w_i be the painted neighbor of w in S_i which is closest to u_i and let w_{i+2} be the neighbor of w in S_{i+2} which is closest to v_{i+1}. (Possibly $w_i=u_i$ or $w_{i+2}=v_{i+1}$). Nodes v_{i+1} and u are now joined by a 3-path configuration:

$P_1=v_{i+1}$, v, v_i, u;

$P_2=v_{i+1}$, the path connecting v_{i+1} to w_{i+2} in S_{i+2}, w, w_i, the path connecting w_i to u_i in S_i, u.

P_3= The path connecting v_{i+1} and u_{i+1} in S_{i+1}, u. ♦

Lemma 2.12 Let (W,v) be an even wheel satisfying Property 2.5 and let u be a Type 4 node. Let P be a subpath of W starting at a neighbor of u in a green sector S_i, ending at a neighbor of u in a blue sector S_j, and containing no other painted neighbors of u. Then

(a) for every Type 2, 3 or 4 node w, strongly adjacent to (W,v), there exists an intermediate node of P in the set $W^u \cap W^v \cap W^w$.

(b) for every pair of Type 2, 3 or 4 nodes x and y, strongly adjacent to (W,v), there exists an intermediate node of P in $W^x \cap W^y$.

Proof of Part (a): If $|V(P)\cap W^v|=1$, the lemma follows as a consequence of Lemma 2.11. So assume $|V(P)\cap W^v|\geq 3$. We use the notation of Figure 2.4. Property 2.5 implies that u is adjacent to all the nodes in the set $X=(V(P)\cap W^v)\setminus\{v_{i+1}, v_j\}$ and u is adjacent to at most one node in each of the sets $\{v_i, v_{i+1}\}$, $\{v_j, v_{j+1}\}$. Let P_{ij} be the subpath of W having v_i and v_{j+1} as endpoints and containing P. Every Type 4 node w with neighbors of distinct colors in P_{ij} satisfies Part (a) of the lemma, since either Lemma 2.11 applies or, as a consequence of Property 2.5 applied to w and (W,v), node w has a neighbor in X. Therefore we can assume w.l.o.g. that a node w contradicting Part (a) of the lemma does not have any green neighbor in P_{ij} and has exactly two neighbors in every blue sector of P_{ij}. Since $|X|$ is odd, there is a node $v^*\in X$ such that the two subpaths of P from v^* to u_i and u_j

respectively, contain the same number of nodes of X. If w is adjacent to v*, Part (a) of the lemma follows since u is adjacent to v*. We assume now that w is not adjacent to v* and we distinguish two cases:

Case 1 Node w has a painted neighbor in S_j, see Figure 2.4.

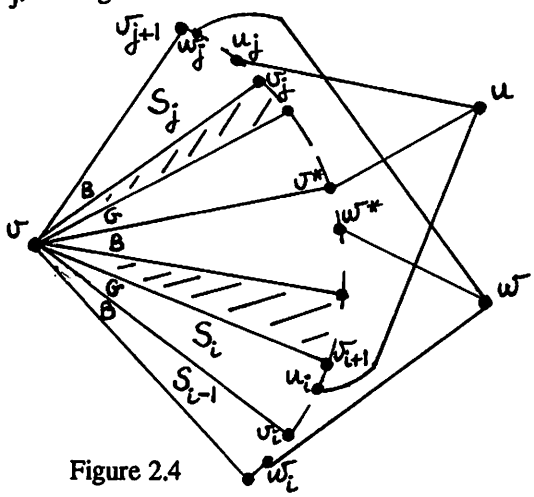


Figure 2.4

Let w_j be the painted neighbor of w, closest to u_j in S_j. We consider the following two subcases:

Case 1.1 Node u is not adjacent to v_i.

Note that S_{i-1} is distinct from S_j, else u has painted neighbors in sectors S_i, S_j, having v_i as common endpoint. Hence by Lemma 2.3 node u is adjacent to v_i.

We assume w.l.o.g. that u_i is the node of N(u) which is closest to v_i in S_i. Let w_i be the node in N(w) closest to v_i in S_{i-1} (possibly $w_i=v_i$). Nodes $v_i \in V^-$ and $u \in V^+$ are connected by the following 3-path configuration:

$P_1=v_i$, v, v*, u.

$P_2=v_i$, the subpath of S_{i-1} connecting v_i and w_i, w, w_j, the subpath of S_j connecting w_j and u_j, u.

$P_3=v_i$, the subpath of S_i connecting v_i and u_i, u.

Case 1.2 Node u is adjacent to v_i but not to v_{i+1}.

The same argument holds, except that node w_i is chosen in S_{i+1}, closest to v_{i+1} and the 3-path configuration connects v_{i+1} and u.

Case 2 Node w has no painted neighbor in S_j, see Figure 2.5.

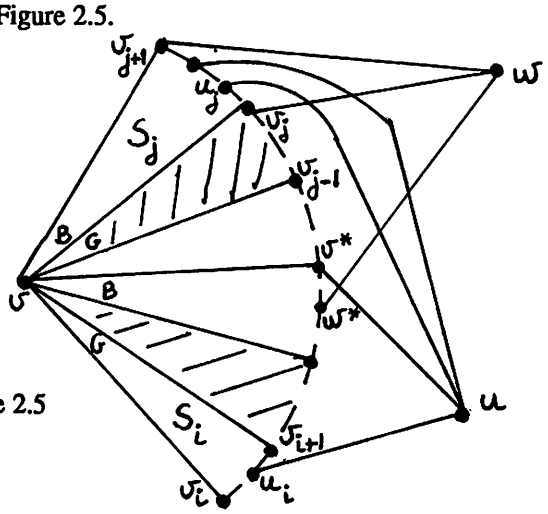


Figure 2.5

This implies that w is adjacent to v_j and v_{j+1}. Since w is not adjacent to v*, w has a painted neighbor, say w*, in the blue sector of (W,v) with endpoint v*. In the subpath P_1 of P connecting v* to v_{j-1}, the sectors of the wheels (W,v) and (W,u) are identical. We divide the proof in the following two subcases:

Case 2.1 Node u is not adjacent to v_j, see Figure 2.5.

In the wheel (W,u), assume w.l.o.g. that the sector containing w* is blue. Then, since (W,u) and (W,v) have the same sectors in P_1, v_j is in a green sector of (W,u). Nodes v_j and w* are neighbors of w and have distinct colors in (W,u). Hence node w is a strongly adjacent node of Type 4 for the wheel (W,u). By Lemma 2.3, node w must have a neighbor in $V(P_1) \cap W^u$, and each of these nodes belongs to W^v. Hence Part (a) follows.

Case 2.2 Node u is adjacent to v_j.

Part (a) follows since v_j is an intermediate node of P in the set $W^u \cap W^v \cap W^w$.

Proof of Part (b): If node x or y coincides with u or v, Part (b) follows directly from Part (a). Assume node x or y, say x, has neighbors painted with distinct colors in $V(P) \cup V(S_i) \cup V(S_j)$. Part (b) now follows by considering the wheel (W,v), having Type 4 node x, and applying Part (a) with u=x and w=y.

Hence both x and y have no neighbors, painted with distinct colors, in $V(P) \cup V(S_i) \cup V(S_j)$. If both x and y satisfy Case 1 of Part (a), then v* is an intermediate node of P in $W^x \cap W^y$. Therefore we can assume w.l.o.g. that node x satisfies Case 2 , hence x is ajacent to v_j, v_{j+1}, but not to v*.

Assume node u is not adjacent to v_j, see Figure 2.5, where x=w. Let w* be the neighbor of x closest to v* in the blue sector of (W,v) having v* as endpoint. Then v_j and w* are neighbors of x and are painted with distinct colors in (W,u).Remark that the subpath of W, having enpoints v_j and w* and containing v*, is contained in P. By applying Part (a) to the wheel (W,u), with Type 4 node x, Part (b) follows.

Finally assume that node u is adjacent to v_j. This implies that u is not adjacent to v_{j+1}. Hence x has neighbors w* and v_{j+1}, painted with distinct colors in (W,u). Remark that in the subpath connecting v_{j+1} to w* and containing S_j, all the nodes adjacent to both u and v are intermediate in this subpath. Part (b) now follows by applying Part (a) to the wheel (W,u), having x as Type 4 node. ♦

Lemma 2.13 Let (W,v) be an even wheel satisfying Property 2.5 and assume that a Type 4 node exists. Then A(W,v) contains all Type 2, 3 and 4 nodes and $\left| \bigcap_{u \in A(W,v)} W^u \right| \geq 2$.

Proof: Every Type 2 node belongs to A(W,v) by definition and Lemma 2.3 shows that every Type 4 node is in A(W,v). Let u be a type 4 node. Then there clearly exist two subpaths, say P_1 and P_2 of W, connecting nodes of W^u painted with distinct colors but containing no other painted node in W^u. Note that P_1, P_2 have no common intermediate node but may have common endnodes. Let Q_1 and Q_2 be the sets of intermediate nodes in P_1, P_2 respectively. Let w be a Type 3 node. Lemma 2.12(a) shows that $|Q_1 \cap W^v \cap W^w| \geq 1$. The same argument applied to Q_2, shows $|W^v \cap W^w| \geq 2$. Hence node w belongs to A(W,v).

To prove the second statement , we apply twice the following argument, first with $Q_i=Q_1$, then with $Q_i=Q_2$.

We make use of the following theorem, due to Helly, see Berge [2].

Let $F=\{F_i, i\in I\}$ be a family of subsets of a finite set. If the set $\cap_{i\in I}F_i$ is empty, then one of the following alternatives holds:

(i) there exist two sets F_i, F_j such that $F_i\cap F_j=\emptyset$,

(ii) there exist three sets F_i, F_j, F_k such that $F_i\cap F_j\neq\emptyset$, $F_i\cap F_k\neq\emptyset$, $F_j\cap F_k\neq\emptyset$ and $F_i\cap F_j\cap F_k=\emptyset$.

Consider the family $F=\{W^u\cap Q_i, u\in A(W,v)\}$. If $\cap_{u\in A(W,v)}W^u\cap Q_i=\emptyset$, then by the Helly theorem, alternative (i) or (ii) must hold for the family F. By Lemma 2.12(b), alternative (i) does not hold. So there must exist three nodes u_1, u_2, u_3 such that $W^{u1}\cap W^{u2}\cap W^{u3}=\emptyset$ and three nodes $v_1\in W^{u1}\cap W^{u2}$, $v_2\in W^{u2}\cap W^{u3}$, $v_3\in W^{u1}\cap W^{u3}$. But then the nodes $u_1, v_1, u_2, v_2, u_3, v_3$ induce a 6-hole. This shows that $\cap_{u\in A(W,v)}W^u\cap Q_i$ is nonempty. Hence we have $|\cap_{u\in A(W,v)}W^u\cap Q_1|\geq 1$ and $|\cap_{u\in A(W,v)}W^u\cap Q_2|\geq 1$. This implies that $|\cap_{u\in A(W,v)}W^u|\geq 2$. ♦

We now examine the structure of the Type 3 nodes and discuss how they relate to the other strongly adjacent nodes in V^+. We continue to assume that the even wheel (W,v) satifies Property 2.5.

Let u and w be two nodes each having more than two neighbors in W. We say that u and w are *weakly nested* if, in every sector S of (W,v) where each of the nodes u and w has two neighbors, either the neighbors of w both belong to the path connecting the two neighbors of u, or vice-versa, the neighbors of u both belong to the path connecting the two neighbors of w. See Figure 2.6 for an example. We say that a family of nodes has the *weak nestedness property* if every pair u, w of nodes in the family is weakly nested.

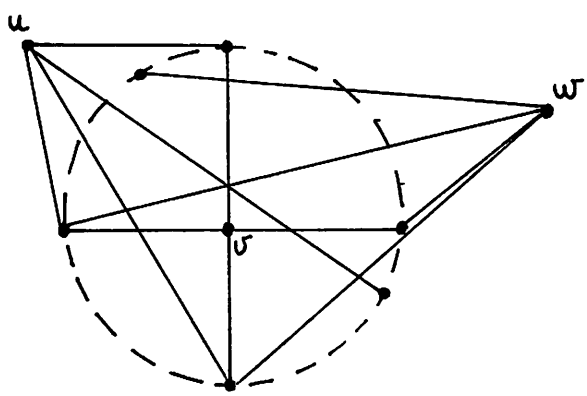


Figure 2.6 Weakly nested nodes u and w.

Lemma 2.14 Let (W,v) be an even wheel satisfying Property 2.5. Then the weak nestedness property holds for the family comprised of the Type 2, 3 and 4 nodes.

Proof: It follows from the definition that v is weakly nested with all other nodes. Assume that two nodes $u,w\neq v$, each having more than two neighbors in W, are not weakly nested. There are two ways in which this could happen.

Case 1 There is a sector S where the neighbors of u, say u_1 and u_2, and the neighbors of w, say w_1 and w_2, appear in the order u_1, w_1, u_2, w_2 where $u_1 \neq w_1$, $u_2 \neq w_1$ and $u_2 \neq w_2$, see Figure 2.7.

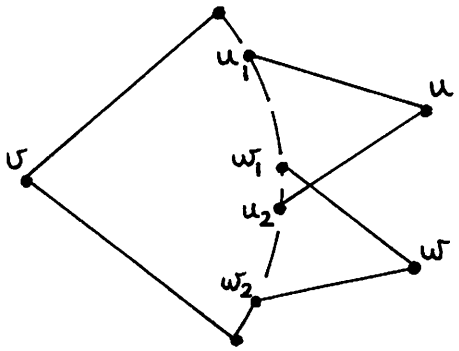


Figure 2.7

In the wheel (W,u), node w has a unique neighbor, namely w_1, in one sector and w_1 is painted. This contradicts Lemma 2.4.

Case 2 There is a sector S_i where the neighbors of u, say u_1 and u_2, and the neighbors of w, say w_1 and w_2, appear in the order u_1, u_2, w_1, w_2 (possibly $u_2 = w_1$), see Figure 2.8.

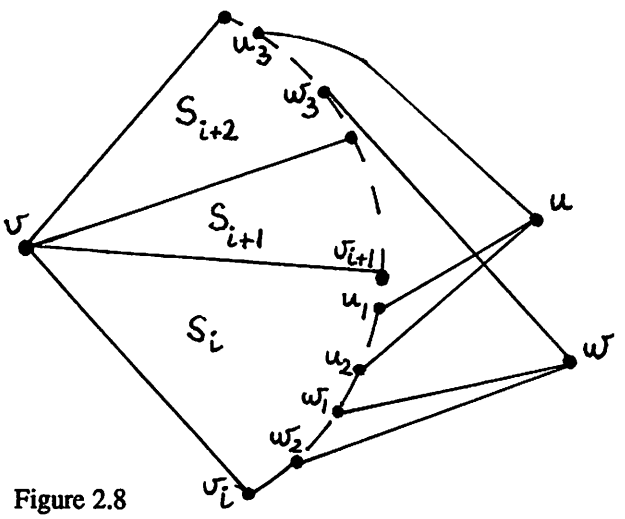


Figure 2.8

With the notation of Figure 2.8, let u_3 be the neighbor of u in $(V(S_{i+1}) \setminus \{v_{i+1}\}) \cup V(S_{i+2})$ which is closest to v_{i+1}, in the path defined by S_{i+1} and S_{i+2}. Similarly, let w_3 be the neighbor of w in $(V(S_{i+1}) \setminus \{v_{i+1}\}) \cup V(S_{i+2})$ closest to v_{i+1}. The nodes u_3 and w_3 exist as a consequence of Property 2.5. Since w is not adjacent to y, it follows from Lemma 2.3 that $w_3 \in V(S_{i+2})$. As a consequence of Property 2.5, node u can have at most two neighbors in a sector of (W,w). This implies that u_3 does not belong to the subpath of W connecting w_1 to w_3 and containing S_{i+1}. Define the hole W' as follows. It contains the edge (u,u_2), the subpath of S connecting u_2 to v_i, the edges (v_i,v), (v,v_{i+2}), the subpath of S_{i+2} from v_{i+2} to u_3 and finally the edge (u_3,u). W' is shorter than W and node w has at least three neighbors in W, contraddicting the assumption that (W,v) satisfies Property 2.5. ♦

Let $u,w \in V^+$ each have more than two neighbors in W. We say that u>w relative to the wheel (W,v) if, in every sector S of (W,v) where each of the nodes u and w has two neighbors, the neighbors of w both belong to the path connecting the two neighbors of u in S. We say that a family of nodes has the *nestedness property* if , for every pair u, w of such nodes, either u>w or w>u or both.

Lemma 2.15 Let (W,v) be an even wheel satisfying Property 2.5. If neither w>u nor u>w holds relative to (W,v), then w is a Type 4 node relative to the wheel (W,u) and u is Type 4 relative to (W,w).

Proof: By Lemma 2.14, nodes u and w are weakly nested. This implies that, if neither w>u nor u>w holds, then there exist two sectors S_i, S_j in which the nodes u, w have their neighbors as in Figure 2.9 where, possibly, either $u_{i1}=w_{i1}$ or $u_{i2}=w_{i2}$ but not both, and, possibly, either $u_{k1}=w_{k1}$ or $u_{k2}=w_{k2}$ but not both. However, since the subpaths of W connecting u_{i1} to u_{k2} (u_{k1} to u_{i2}) and not containing u_{i2} (u_{i1}) have an even number of nodes in W^v, it follows that, in each set $\{w_{k1},w_{k2}\}$, $\{w_{i1},w_{i2}\}$ at least one of the nodes is painted in (W,u) and the colors are distinct. The proof for u and (W,w) is identical. ♦

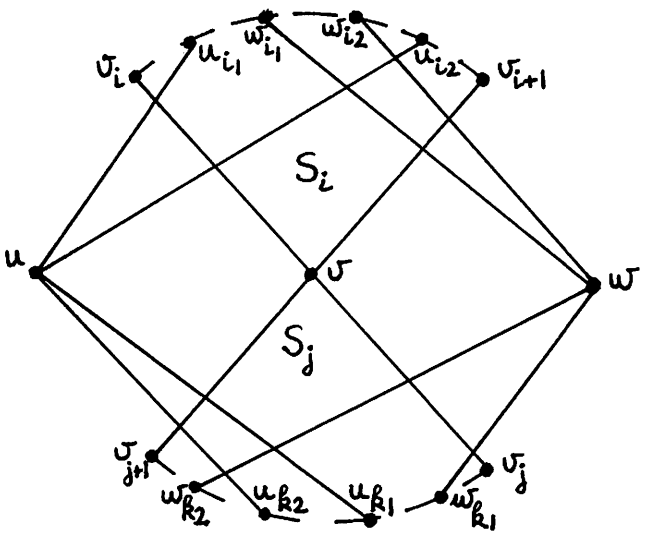


Figure 2.9

We are now ready to prove the main result of this section.

Proof of Theorem 2.2 If there exists an even wheel satisfying Property 2.5 and having at least one Type 4 node, then the result holds as a consequence of Lemma 2.13. So, we now consider the case where every even wheel satisfying Property 2.5 has no Type 4 node.

Let (W,v) be an even wheel satisfying Property 2.5 and let B denote the set of nodes of V^+ with more than two neighbors in W. Lemma 2.15 shows that the nestedness property holds for the family B. Consider the family *F* comprising the Type 2 nodes and Type 3 nodes that have painted neighbors in blue sectors. The definition of the relation > implies that, if $x,y,z \in F$ satisfy x>y and y>z, then we also have x>z, i.e. the relation > is transitive in the family *F*. (This follows from the observations that a contradiction to x>z must occur in a blue sector and that y

also has two neighbors in that sector). Therefore, we can find $u^* \in F$ such that $x > u^*$ for every $x \in F$. Now, consider the wheel (W,u^*). Note that Property 2.5 is satisfied by (W,u^*) and, w.l.o.g., its blue sectors are subpaths of the blue sectors of (W,v). By our choice of u^*, no node of B has a blue neighbor in (W,u^*). By Lemma 2.15, B is a nested family relative to (W,u^*). Since all Type 3 nodes relative to (W,u^*) have green neighbors, the relation $>$ is transitive on B. Recall that the definition of nestedness also implies that, for $x,y \in B$ either $x>y$ or $y>x$ or both. So we have a total transitive order on B.

For $i \geq 1$ integer, let $A_i(W,u^*)$ be the set of Type 2 and 3 nodes w, strongly adjacent to (W,u^*) such that $|W^w \cap W^{u^*}| \geq i$. The total transitive order of the relation $>$ implies that $|\cap_{u \in A_i(W,u^*)} W^u| \geq i$. Note that the set $A(W,u^*)$ of Theorem 2.2 coincides with $A_2(W,u^*)$. Hence Theorem 2.2 holds for the wheel (W,u^*) and the proof is complete. ♦

Since the wheels used in the proof of Theorem 2.2 satisfy Property 2.5, we make the following observation.

Remark 2.16 There exists an even wheel (W,v) with Property 2.5 that satisfies Theorem 2.2.

3. A DOUBLE STAR CUTSET THEOREM FOR SHORT WHEELS

In this section, we prove the following key result concerning the decomposition of balanced bipartite graphs that contain an even wheel.

Theorem 3.1 Let (W,v) be an even wheel satisfying Property 2.5 in a balanced bipartite graph. Then every path connecting a blue node to a green node of (W,v) contains a node in $N(v) \cup A(W,v)$.

Corollary 3.2 A balanced bipartite graph containing an even wheel has a double star cutset.

Proof: There exists an even wheel (W,v) satisfying Property 2.5 and such that $\cap_{u \in A(W,v)} W^u \neq \emptyset$, as a consequence of Theorem 2.2 and Remark 2.16. Now, for any $f \in \cap_{u \in A(W,v)} W^u$, Theorem 3.1 implies that $N(v) \cup N(f)$ is a double star cutset. ♦

A bipartite graph is *linear* if the corresponding incidence matrix is linear, as defined in the introduction. Therefore, a bipartite graph G is linear if and only if it contains no cycle of length 4. Linear balanced bipartite graphs have been studied by Conforti and Rao [8], who have proved the following (single) star cutset theorem. This theorem can now be deduced from Theorem 3.1.

Corollary 3.3 A linear balanced bipartite graph containing an even wheel has a star cutset.

Proof: Let (W,v) be an even wheel in a linear balanced bipartite graph. Then we have $A(W,v)=\{v\}$. Otherwise, let u be some other node in A(W,v) and let f_1 and f_2 be two neighbors of u in W^v. The nodes u, v, f_1 and f_2 induce a cycle of length 4, contradicting the linearity assumption. Now let (W,v) be an even wheel satisfying Property 2.5. It follows from Theorem 3.1 that N(v) is a star cutset. ♦

In the proof of Theorem 3.1, we make use of the following lemma, which appears in [8].

Lemma 3.4 Let (W,v) be an even wheel in a balanced bipartite graph G and let P be a chordless path with nodes in $V(G)\setminus(V(W)\cup N(v))$ such that any $x\in V(P)$ is adjacent to at most one node of W^v and to no painted node of W. Then at most two nodes of W^v have at least one neighbor in V(P).

Proof: Assume the lemma is not true and let P' be a shortest subpath of P with the property that three distinct nodes of W^v have at least one neighbor in V(P'). Denote by v_1, v_2, v_3 the three nodes of W^v with at least one neighbor in V(P'). Let P_{ij} be the subpath of W connecting v_i to v_j and not containing v_k, for $i,j,k\in\{1,2,3\}$ and $i\neq j\neq k$. Since W is a par hole, at least one of the three paths P_{ij} has length 0 (mod 4). Assume w.l.o.g. that P_{12} has length 0 (mod 4). Let $u_1\in N(v_1)\cap V(P')$ and $u_2\in N(v_2)\cap V(P')$ be chosen such that the subpath R of P' connecting them is shortest. Let H be the hole formed by P_{ij} and R. Then v has an odd number, greater than one, of neighbors in H. Hence (H,v) is an odd wheel. ♦

Proof of Theorem 3.1: If the theorem does not hold for (W,v), let $P=(s^*,s,...,t,t^*)$ be a shortest path connecting nodes of W with distinct colors, and containing no node of $N(v)\cup A(W,v)$. W.l.o.g. assume that $v\in V^+$, s^* is green and t^* is blue. The following possibilities can occur for nodes s and t.

(a) Node s (or t) has only one neighbor in W, namely s^* (t^* resp.),

(b) Node s (or t) belongs to $V^-\setminus N(v)$, is strongly adjacent to W, but all its neighbors are in the same sector of (W,v),

(c) Node s (or t) belongs to $V^-\setminus N(v)$ and has exactly two neighbors in (W,v), one in sector S_i and one in sector S_j, $i\neq j$, where S_i and S_j have the same color,

(d) Node s (or t) belongs to $V^+\setminus A(W,v)$ and is a Type 1 node,

(e) Node s (or t) belongs to $V^+\setminus A(W,v)$ and is a Type 3 node with at most one neighbor in W^v.

It follows from Theorem 2.1 and Lemma 2.3 that no other possibility can occur for the node s (or t).

Next, we show that we can dispose of the possibilities (b) and (d) by modifying the wheel (W,v) and the path P.

Claim 1: There exists a wheel (W',v) and a path $P'=(s^{*'},s',...,t',t^{*'})$ connecting nodes of distinct colors in (W',v), containing no node of $N(v)\cup A(W',v)$, such that the nodes s' and t' satisfy one of the properties (a), (c) or (e) above and, furthermore, the nodes of $V(P')\setminus\{s^{*'},s',t',t^{*'}\}$ have at most one neighbor in W'.

Proof: First, assume that some node u of $V(P)\setminus\{s^*,s,t,t^*\}$ has at least two neighbors in W. These neighbors are unpainted, otherwise a shorter path P would exist. All Type 2 nodes are in A(W,v), so u must be of Type 1. Denote by v_i and v_{i-1} the two nodes of W adjacent to u and by S_i the sector of (W,v) with endpoints v_i and v_{i-1} and assume w.l.o.g. that S_i is a blue sector. Construct W' from W by replacing the sector S_i by the sector (v_{i-1},u,v_i) and let P' be the subpath of P connecting s* to u. Note that A(W',v)=A(W,v). Therefore, P' connects sectors of distinct colors in (W',v) and contains no node of $N(v)\cup A(W',v)$. In P', the node t' adjacent to u is different from s (if s=t', then this node is Type 4 relative to (W',v) but all Type 4 nodes belong to A(W',v).) Note also that P' is shorter than P. So, by repeating the above procedure, we can dispose of all the nodes of $V(P)\setminus\{s^*,s,t,t^*\}$ with at least two neighbors in W. In the remainder, we assume w.l.o.g. that the nodes of $V(P)\setminus\{s^*,s,t,t^*\}$ have at most one neighbor in W and, if this neighbor exists, it is unpainted.

Assume that s satisfies property (b) or (d) and let S_i be the sector containing s*. Denote by v_i and v_{i-1} the endpoints of S_i and by s_i and s_{i-1} the neighbors of s in S_i that are closest to v_i and v_{i-1} respectively. Let (W',v) be the wheel obtained from (W,v) by substituting the subpath of S_i connecting s_{i-1} to s_i with (s_{i-1},s,s_i) and let P' be the subpath obtained from P by removing the node s*, namely P'=(s,s',...,t,t*). Since A(W',v)=A(W,v), the path P' connects two sectors of (W',v) with distinct colors and contains no node of $N(v)\cup A(W',v)$. Note that s'=t cannot occur, since this node would be Type 4 relative to (W',v), a contradiction to the fact that P' contains no node of A(W',v). Therefore, s' has at most two neighbors in W'. If s' does have two neighbors, it must be of Type 1 relative to (W',v), i.e. property (d) holds. In this case the above procedure can be repeated and P' can be shortened again. ♦

As a consequence of this claim, we can assume w.l.o.g. that (W,v) and P=(s*,s,...,t,t*) have the following properties, in addition to those already stated at the beginning of the proof: s and t satisfy Properties (a), (c) or (e) and the nodes of $V(P)\setminus\{s^*,s,t,t^*\}$ have at most one neighbor in W.

Claim 2: Let s be a Type 3 node.
(i) If $W^S\cap W^V=\emptyset$, then no node of $V(P)\setminus\{s^*,s,t,t^*\}$ is adjacent to a node of W.
(ii) If $W^S\cap W^V=\{f\}$, then no node of $V(P)\setminus\{s^*,s,t,t^*\}$ is adjacent to a node of $W\setminus\{f\}$.

Proof: Assume not and let u be the node of $V(P)\setminus\{s^*,s,t,t^*\}$ which is closest to s in P and adjacent to a node of W (case (i)) or of $W\setminus\{f\}$ (case(ii)). By Claim 1, node u can only be adjacent to one node of W and this node is unpainted. Let $x\in W^V$ be this node, see Figure 3.1.
By Remark 2.9, node s has exactly two neighbors in each green sector of (W,v). By Property (e), s has at most one unpainted neighbor in W. Let S_i be the green sector having x as endpoint and let s_i be the neighbor of s closest to x in S_i. Let S_j be a green sector distinct from S_i, say with endpoints v_j and v_{j-1} and let s_j and s_{j-1} be the neighbors of s in S_j, closest to v_j and v_{j-1} respectively. Assume w.l.o.g. that s_j is painted. Then $x\in V^-$ and $s\in V^+$ are connected by a 3-path configuration:
P_1=x, u, the subpath of P connecting u to s
P_2=x, the subpath of S_i connecting x to s_i, s

P_3=x, v, v_j, the subpath of S_j connecting v_j to s_j, s. ♦

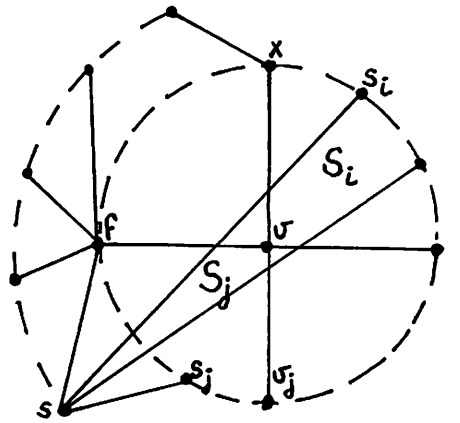


Figure 3.1

A similar statement to Claim 2 holds when t is of Type 3.

Claim 3: Neither node s nor node t is of Type 3.

Proof: Assume s is a Type 3 node. By Remark 2.9, node s has exactly two neighbors in each green sector of (W,v). Furthermore s has at most one unpainted neighbor, say f. Let S_i be the blue sector containing t*. We choose an adjacent green sector S_{i+1} as follows, see Figure 3.2.

(i) If S_i has node f as an endpoint, let S_{i+1} be the green sector, adjacent to S_i, which does not contain f as endpoint.

(ii) If S_i does not have node f as an endpoint but one of the adjacent green sectors has f as endpoint, let S_{i+1} be the green sector having f as endpoint.

(iii) If neither of the sectors adjacent to S_i has f as an endpoint (or if f does not exist), choose S_{i+1} arbitrarily to be one of the sectors adjacent to S_i.

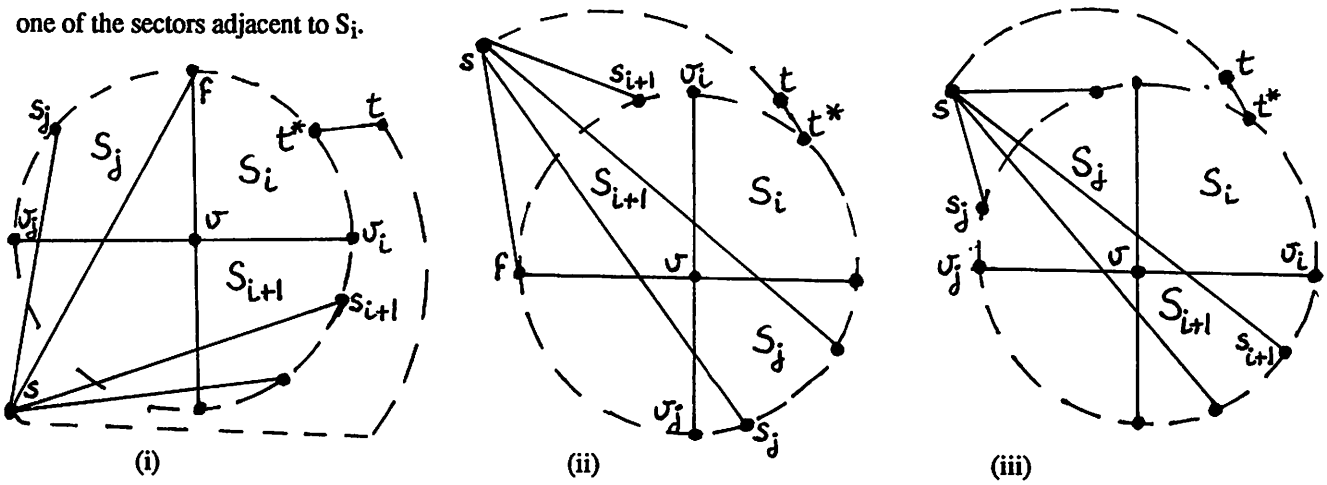


Figure 3.2

Let v_i be the common endpoint to S_i and S_{i+1} and let s_{i+1} be the neighbor of s in S_{i+1} which is closest to v_i. Note that s_{i+1} is a painted node since $v_i \neq f$. Finally, let S_j be the green sector, distinct from S_{i+1} and adjacent to S_i. Let $v_j \neq f$ be an endpoint of S_j which is not adjacent to t* and let v_{j-1} be the other endpoint of S_j. (Note that such a choice of v_j is always possible due to the above conditions (i)-(iii).) Let s_j be the neighbor of s in S_j which is closest to v_j.

Case 1: Node t is not adjacent to v_j.

Now the nodes $v_i \in V^-$ and $s \in V^+$ are connected by a 3-path configuration:

$P_1 = v_i$, the subpath of S_i connecting v_i to t*, the subpath of P from t* to s.

$P_2 = v_i$, the subpath of S_{i+1} connecting v_i to s_{i+1}, s.

$P_3 = v_i$, v, v_j, the subpath of S_j from v_j to s_j, s.

Case 2: Node t is adjacent to v_j.

This implies that t is a Type 3 node and that no node of $V(P)\backslash\{s^*,s,t,t^*\}$ has a neighbor in W^v, else s or t contradicts Claim 2. Let P* be the subpath of P having s and t as endnodes. Then $v_i \in V^-$ and $s \in V^+$ are connected by a 3-path configuration.

$P_1 = v_j$, t, P*, s.

$P_2 = v_j$, the subpath of S_j from v_j to s_j, s.

In case (i) or (ii), P_3 is chosen as follows:

$P_3 = v_j$, v, f, s.

In case (iii), let s_{j-1} be the neighbor of s closest to v_{j-1} in S_j. Then P_3 is chosen as follows:

$P_3 = v_j$, v, s_{j-1}, the subpath of S_j from v_{j-1} to s_{j-1}, s. ♦

Claim 4: If s satisfies Property (c), then there exists a green sector S with the property that each endpoint of S is adjacent to at least one node in the set $V(P)\backslash\{s^*,s,t,t^*\}$.

Proof: Since node s satisfies Property (c), Theorem 2.1 implies that there exists a sector S_i, say with endpoints v_i and v_{i-1}, such that the unique neighbor s_i of s in S_i is not adjacent to v_i, v_{i-1}. Let S_j be the sector containing the second neighbor s_j of s and let v_j, v_{j-1} be the endpoints of S_j. By Lemma 3.4, at most two nodes of W^v are adjacent to $V(P)\backslash\{s^*,s,t,t^*\}$.

Case 1: No node of W^v is adjacent to $V(P)\backslash\{s^*,s,t,t^*\}$.

Let S_k be the sector containing t* and let $v_k \neq v_j, v_{j-1}$ be an endpoint of S_k. We can assume w.l.o.g. that $v_i \neq v_k$ and that v_j is not adjacent to t*. Then, $v \in V^+$ and $s \in V^-$ are connected by a 3-path configuration:

$P_1 = v$, v_k, the subpath of S_k from v_k to t*, P

$P_2 = v$, v_i, the subpath of S_i from v_i to s_i, s

$P_3 = v$, v_j, the subpath of S_j from v_j to s_j, s.

Case 2: Exactly one node of W^v, say v_l, is adjacent to $V(P)\backslash\{s^*,s,t,t^*\}$.

Starting from s, let u* be the first neighbor of v_l encountered on P.

Case 2.1: $v_l \neq v_j, v_{j-1}$.

Assume w.l.o.g. that $v_l \neq v_i$. Then v and s are connected by a 3-path configuration:

P_1=v, v_l, u*, the subpath of P from u* to s,

P_2 and P_3 are the same as in Case 1.

Case 2.2: $v_l = v_{j-1}$ and t satisfies Property (a).

Then, the 3-path configuration of Case 1 is still valid, except if t* is adjacent to v_j. So, we consider the case where t* is adjacent to v_j. Let Q be the path of W connecting v_{j-1} to t* and not containing v_j. Let v* be the neighbor of v_{j-1} which is closest to t* on P and let P* be the subpath of P connecting t* to v*. Let H be the hole formed by Q, P* and the edge (v_{j-1},v^*). Then (H,v) is an odd wheel.

Case 2.3: $v_l = v_{j-1}$ and t satisfies Property (c).

As a consequence of Theorem 2.1, one of the neighbors of t in W is adjacent to no node of W^V. Choose t* to be such a neighbor of t. Then the argument of Case 1 still holds.

Case 3: Two nodes of W^V are adjacent to $V(P)\backslash\{s^*,s,t,t^*\}$.

Starting from s, let u* be the first node of P having a neighbor in W^V, say $v_l \in W^V$. If $v_l \neq v_j, v_{j-1}$, then the argument of Case 2.1 still holds. So, assume w.l.o.g. that $v_l = v_{j-1}$. Let v_p be the other node of W^V with neighbors in $V(P)\backslash\{s^*,s,t,t^*\}$. Starting from s, let w* be the first neighbor of v_p encountered on P. Assume w.l.o.g. that $v_i \neq v_p$. If $v_p \neq v_j$, then the nodes v and s are connected by 3-path configuration:

P_1=v, v_p, w*, the subpath of P from w* to s. P_2 and P_3 are as in Case 1.

Hence v_p and v_l are the endpoints of the green sector S_j and the claim follows. ♦

If both s and t satisfy Property (c), then Claim 4 implies that at least three nodes of W^V, namely the endpoints of two sectors of distinct colors, have neighbors in $V(P)\backslash\{s^*,s,t,t^*\}$. This contradicts Lemma 3.4 asserting that at most two nodes of W^V can have neighbors in $V(P)\backslash\{s^*,s,t,t^*\}$. So we can assume w.l.o.g. that t satisfies Property (a). The next claim shows that this cannot occur either, proving the theorem.

Claim 5: Node t cannot satisfy Property (a).

Proof: Assume t satisfies Property (a) and let v_j, v_{j-1} be the endpoints of the sector S_j containing t*.

First, we show that at least one node of W^V has a neighbor in $V(P)\backslash\{s^*,s,t,t^*\}$. Assume not. Then Claim 4 implies that node s satisfies Property (a). Let P_1 and P_2 be the two paths of W connecting s* to t*. Let H_1 (H_2) be the hole formed by P and P_1 (P_2 respectively). Both H_1 and H_2 have an odd number of neighbors of v and, for at least one of the holes this number is greater than one. So either (H_1,v) or (H_2,v) is an odd wheel.

When traversing P from t*, let u* be the first node encountered which has a neighbor v_l in W^V. We show that $v_l = v_j$ or v_{j-1}. Assume not and let P_1 and P_2 be the paths of W connecting t* to v_j and let P* be the subpath of P connecting t* to u*. Let H_1, H_2 be the two holes formed by P*, the edge (u^*,v_j) and P_1, P_2 respectively. Note that v has and odd number of neighbors in one of these two holes. This odd number is greater than one, since $v_l \neq v_j, v_{j-1}$. Hence, either (H_1,v) or (H_2,v) is an odd wheel. So v_l is an endpoint of S_j. Assume w.l.o.g. that $v_l = v_j$.

If v_j is the only node of W^v with neighbors in $V(P)\backslash\{s^*,s,t,t^*\}$ then, by Claim 4, node s satisfies Property (a) and, therefore, v_j must also be an endpoint of the sector containing s^*. In other words, s^* and t^* belong to adjacent sectors. Let H be the hole formed by P and the path of W connecting s^* to t^* which does not contain v_j. Then (H,v) is an odd wheel.

So there must be a second node of W, say $v_i \neq v_j$, with neighbors in $V(P)\backslash\{s^*,s,t,t^*\}$. Let w be the neighbor of v_i in P which is the closest to t^* and let Q be the subpath of P connecting w to t^*. Let w^* be the neighbor of v_j closest to w in Q, and let Q^* be the subpath of Q connecting w to w^*. Let P_1 be the subpath of W connecting v_i to v_j and which does not contain t^*. Let P_2 be the subpath of W connecting v_i to t^* and which does not contain v_j. Finally, define the holes H_1 and H_2 as follows. H_1 is formed by P_1, Q^* and the edges (w,v_i), (w^*,v_j). H_2 is formed by P_2, Q and the edge (w,v_i). One of the holes H_1, H_2 contains an odd number of neighbors of v and, if v_i is not an endpoint of the sector containing t^*, the number of neighbors of v is greater than one in each of the holes H_1 and H_2. So, either (H_1,v) or (H_2,v) is an odd wheel. So $v_i=v_{j-1}$.

If s satisfies Property (a), then the same argument shows that v_{j-1} and v_j are also the endpoints of the sector containing s^*, a contradiction to the fact that s^* and t^* are in different sectors. If s satisfies Property (c), then, by Claim 4, v_{j-1} and v_j are the endpoints of a green sector, a contradiction to the fact the the sector containing t^* is painted blue. ♦

REFERENCES

[1] C. Berge, "Balanced Matrices," *Mathematical Programming 2* (1972), 19-31.

[2] C. Berge, *Graphs and Hypergraphs*, North Holland (1973).

[3] O. Carducci, Ph.D. Thesis, Mathematics department, Carnegie Mellon University, Pittsburgh, Pa. (1989).

[4] F.G. Commoner, "A Sufficient Condition for a Matrix to be Totally Unimodular,"*Networks 3* (1973), 351-365.

[5] M. Conforti and G. Cornuéjols, "An Algorithmic Framework for the Matching Problem in Some Hypergraphs," *Networks 17* (1987), 365-386.

[6] M. Conforti, G. Cornuéjols and M.R. Rao, "Balanced Matrices," forthcoming.

[7] M. Conforti and M.R. Rao, "Structural Properties and Decomposition of Restricted and Strongly Unimodular Matrices," *Mathematical Programming 38* (1987), 17-27.

[8] M. Conforti and M.R. Rao, "Structural Properties and Decomposition of Linear Balanced Matrices," (1988), to appear in *Mathematical Programming*.

[9] M. Conforti and M.R. Rao, "Odd Cycles and Matrices with Integrality Properties," *Mathematical Programming 45* (1989), 279-294.

[10] D.R. Fulkerson, A.J. Hoffman and R. Oppenheim, "On Balanced Matrices,"*Mathematical Programming Study 1* (1974), 120-132.

[11] K. Truemper, "Alpha-Balanced Graphs and Matrices and GF(3)-Representability of Matroids," *Journal of Combinatorial Theory B 32* (1982), 112-139.

[12] M. Yannakakis, "On a Class of Totally Unimodular Matrices," *Mathematics of Operations Research 10* (1985), 280-304.

NEW RESULTS ON FACETS OF THE CUT CONE(°)

Michel Deza(*) and Monique Laurent(**)

Abstract

The cut cone C_n (cut polytope P_n) is the conic (convex) hull of the 2^{n-1} cuts of the complete graph on n nodes. We describe several new classes of facets within the class of clique-web inequalities introduced in [DL1],[DL2]. Another class of facets, the parachute facet, is shown to be closely related to Fibonacci numbers. By screening several examples, we indicate some methods for the explicit enumeration of the distinct (up to permutation) switchings of a given facet, permutation and switching being specific operations on the facets of C_n,P_n.

(*) CNRS, Université Paris 7
17, Passage de l'industrie, 75010 Paris, France
(**) CNRS, LAMSADE, Université Paris Dauphine
Place du Maréchal de Lattre de Tassigny
75775 Paris cedex 16, France
(°) This work was partially supported by RISE (Research Institute of Software Engineering Foundation), Tokyo.

0. INTRODUCTION

Consider the complete graph K_n on nodeset [1,n] = {1,2,..,n}. Given a subset S of [1,n], the **cut** determined by S is the set of edges of K_n having exactly one endnode in S; its incidence vector is the vector, denoted as $\delta(S)$ and called cut vector determined by S, in $R^{n(n-1)/2}$ defined by: $\delta_{ij}(S)=1$ if $i\in S$, $j\notin S$, or $i\notin S$, $j\in S$, and $\delta(S)_{ij}=0$ otherwise, for $1\le i<j\le n$. So, $\delta(S)=\delta([1,n]-S)$ and there are 2^{n-1} distinct cut vectors. For simplicity, we shall use the same notation $\delta(S)$ for denoting the cut both as edgeset or its incidence vector. The cut cone C_n is the polyhedral cone in $R^{n(n-1)/2}$ generated by all cut vectors of K_n while the cut polytope P_n is the convex hull of all cut vectors. Both polyhedra P_n, C_n are full dimensional. Hence any facet of P_n is uniquely (up to positive multiple) defined by an inequality of the form: $v.x\le v_0$, the cuts $\delta(S)$ satisfying $v.\delta(S)=v_0$ are called the **roots** of the facet; the facets of C_n correspond to the facets of P_n supported by an inequality of the form: $v.x\le 0$. Two operations on facets of P_n, C_n are known: the trivial permutation operation and the switching operation, switching in the cut cone C_n was introduced in [D2] and in the cut polytope (of arbitrary graphs) in [BM]. In fact, permutation and switching are the only symmetries of the cut polytope P_n ([DGriL]). A remarkable feature of the cut polytope is that all its facets can be obtained -via the switching operation- from the facets of the cut cone ([BM]). Therefore, from the facial structure point of view, it is enough to consider the cut cone C_n. Actually the study of facets of the cut cone C_n applies also directly to the k-cut polytope, i.e. the convex hull of all h-cuts for $2\le h\le k$ (k-cuts are the edgesets cut by a partition of the nodeset into k parts), since any facet of C_n also defines a (homogeneous) facet of the k-cut polytope ([DGröL]).

In this paper, we give the following three types of results on the facial structure of the cut cone. In the first section, we describe several new classes of facets within the known class

of clique-web inequalities introduced in [DL2]. The other class of facets of C_n, the parachute facet, considered in the second section, turns out to be related to Fibonacci sequences. In the last section, by screening several examples of facets from sections 2-3, we indicate some methods for the explicit enumeration of the distinct, up to permutation, switchings of a given facet.

The graph notations and the notions of polyhedral combinatorics that we use are classical. For instance, C(1,2,..,n) (resp. P(1,2,..,n)) is the cycle (resp. the path) with edges (1,2),(2,3),..,(n-1,n),(n,1) (resp. (1,2),..,(n-1,n)). An inequality: $v.x \le v_0$ is called **valid** over a polyhedron P if $P \subseteq \{x: v.x \le v_0\}$ and, then, **facet inducing** inducing over P if the dimension of $\{x \in P: v.x = v_0\}$ is one less the dimension of P. Hence, an inequality $v.x \le v_0$ is facet inducing over C_n if one can find n(n-1)/2 - 1 linearly independent roots. We use below, in particular for the definition of clique-web inequalities, the following notion of **collapsing**. Given $N \ge n$ and a partition π of [1,N] into n classes $I_1,..,I_n$ and a vector v of $R^{N(N-1)/2}$, the π-collapsing of vector v is the vector v_π of $R^{n(n-1)/2}$ defined by: $(v_\pi)_{ij} = \sum_{h \in I_i,\ k \in I_j} v_{hk}$ for $1 \le i < j \le n$.

If G is a weighted graph on nodeset [1,N] with weights $(v_{ij})_{1 \le i < j \le N}$ on its edges, then the vector $v = (v_{ij})_{1 \le i < j \le N}$ is called the **edgeweight vector** of G. The graph obtained by π-collapsing of G is the weighted graph on nodeset [1,n] with edgeweight vector v_π. Given scalars $b_1,..b_n$, we denote by $Q(b_1,..,b_n).x$ the linear form: $\Sigma_{1 \le i < j \le n} b_i b_j x_{ij}$ and given a subset S of [1,n], we set: $b(S) = \Sigma_{i \in S} b_i$.

1. NEW FACETS

In [DL2], we introduced a class of valid inequalities over C_n: the clique-web (or CW) inequalities, denoted by $CW^r_{p,q}(b)$ and defined as follows. Let us recall the notion of collapsed antiweb. First, recall that the **antiweb** AW^r_p is the circular graph on nodeset [1,p] with edges the pairs (i,i+1),(i,i+2),..,(i,i+r) for i=1,..,p (setting p+1=1). In the following, we use the same notation AW^r_p for denoting the antiweb graph and its set of edges. Given integers $b_1,..,b_p > 0$, set $P = \Sigma_{1 \le i \le p} b_i$ and consider the partition $\pi(b)$ of [1,P] into the p intervals $[b_1+..+b_i+1, b_1+..+b_{i+1}]$ for i=0,1,..,p-1; then the **collapsed antiweb** $AW^r_p(b_1,..,b_p)$ is the weighted graph obtained by $\pi(b)$-collapsing of the antiweb AW^r_P.

Definition 1.1. Given integers $p,q \ge 2$, $n=p+q$, $r \ge 0$, $b_1,..b_p > 0$, $b_{p+1},..,b_n < 0$ with $\Sigma_{1 \le i \le n} b_i = 2r+1$, the clique-web inequality $CW^r_{p,q}(b)$ is defined as follows:

$$\sum_{1 \le i < j \le n} b_i b_j x_{ij} \; - \sum_{(i,j) \in AW^r_p(b)} x_{ij} \le 0$$

where $AW^r_p(b) = AW^r_p(b_1,..,b_p)$ is a collapsed antiweb as defined above.

In above definition 1.1, relation $\sum_{(i,j) \in AW^r_p(b)} x_{ij}$ should be understood as $\sum_{1 \le i < j \le p} v_{ij} x_{ij}$ where $v = (v_{ij})_{1 \le i < j \le p}$ is the edgeweight vector of $AW^r_p(b)$. Note that, for r=0, the antiweb $AW^r_p(b)$ is empty graph, so $CW^0_{p,q}(b) = \Sigma\, b_i b_j x_{ij}$ is simply $Q(b).x$ with $\Sigma\, b_i = 1$, also called **hypermetric** inequality and introduced in [D1]. Also, for $r \ge 1$, $CW^r_{2r+3,2}(1,..,1,-1,-1)$ corresponds (via switching) to the **bicycle odd wheel** inequality introduced in [BGM],[BM]. Some classes of CW facets were given in [DL1],[DDL] for r=0,1 and in [DL2],[DGröL] for arbitrary r. In particular, it was shown that $CW^r_{p,q}(b_1,..b_p,-1,..,-1)$ (with $\Sigma_{1 \le i \le p} b_i - q = 2r+1$) is facet inducing

for all $p\geq 5$, $b_1,..,b_p\geq r\geq 1$ ([DGröL],corollary 2.18). Observe that, for $b_1,..,b_p\geq r\geq 1$, the collapsed antiweb $AW^r_p(b)$ is simply: $r(r+1)/2$ $C(1,2,..,p)$, i.e. the cycle on nodes $(1,2,..,p)$ with weights $r(r+1)/2$ on its edges; then, the roots of $CW^r_{p,q}(b_1,..,b_p,-1,..,-1)$ are the cuts $\delta(S\cup S')$ where S is a circular interval of [1,p], S' is a subset of [p+1,n] and $b(S) - |S'| = r$ or $r+1$ ([DL2]). The next result gives the characterization of all facets of the form $CW^r_{p,q}(b_1,..b_p,-1,..,-1)$ with $b_1,..,b_p\geq r\geq 1$.

Theorem 1.2. Given integers $p\geq 3, r\geq 1, q\geq 2$, $n=p+q$, $b_1,..,b_p\geq r$ and $\Sigma_{1\leq i\leq p}\, b_i - q = 2r+1$, the following assertions are equivalent:
(i) $CW^r_{p,q}(b_1,..,b_p,-1,..,-1)$ is facet inducing over C_n
(ii) $p\geq 5$ or ($p=4$ and $b_1,b_2\geq r+1$ -up to cyclic shift on [1,4]-) or ($p=3$ and $b_1\geq r+2$, $b_2,b_3\geq r+1$ - up to cyclic shift on [1,3]-)

We first recall a lifting result for CW inequalities which follows from theorem 2.12,[DGröL] as well as a useful lemma for the proofs of facetness which follows from lemma 2.5, [BGM].

Proposition 1.3 ([DGröL]). Assume that $b_1,..,b_p\geq r$, $b_{p+1},..,b_n<0$ and $b_j=b_k=d$ for some distinct j,k in [p+1,n]. If $CW^r_{p,q}(b_1,..,b_n)$ is facet inducing, then so is $CW^r_{p,q+1}(b_1,..,b_{i-1},b_i-d,b_{i+1},..,b_n,d)$ for any node i of [1,p].

Lemma 1.4 ([BGM]). Given $a\in R^{n(n-1)/2}$, disjoint subsets I,J,H,S of [1,n] such that the cut vectors $\delta(S\cup J)$, $\delta(S\cup H)$, $\delta(S\cup I\cup J)$, $\delta(S\cup I\cup H)$ satisfy equality: $a.x=0$, then $\Sigma_{i\in I\, j\in J}\, a_{ij} = \Sigma_{i\in I\, h\in H}\, a_{ih}$ holds.

Proof of theorem 1.2. We first show that (i) implies (ii). Assume that $v=CW^r_{p,q}(b_1,..,b_p,-1,..,-1)$ is facet inducing with $p=3,4$. Consider first the case: $p=3$. We can suppose for instance that $b_1\geq b_2\geq b_3$. Using the necessary condition for facetness stated in (proposition 2.1, [DL2]), we must have that: $b_1+b_2\leq q+r$, implying that $b_3\geq r+1$. Suppose, for contradiction, that $b_1=r+1$; then the roots of $v=CW^r_{p,q}(r+1,r+1,r+1,-1,..,-1)$ are the cuts $\delta(\{1\})$, $\delta(\{2\})$, $\delta(\{3\})$, $\delta(\{1,i\})$, $\delta(\{2,i\})$, $\delta(\{3,i\})$ for $i\in[4,n]$ and hence they are all roots of inequality $CW^r_{p,q}(1,1,1,-1,-1,0,0,..,0)$ ($p=3,q=r+2$). Therefore, v is dominated by a valid inequality and thus is not facet inducing. Consider now the case: $p=4$. One can check easily that the roots of each of the following inequalities: $CW^r_{4,x+r-1}(x,r,r,r,-1,..,-1)$ for $x\geq r$, $CW^r_{4,x+y-1}(x,r,y,r,-1,..,-1)$ for $x,y\geq r+1$ are, in fact, also roots of $CW^0_{2,n-2}(1,0,1,0,-1,0,..,0)$ (putting appropriate number n-1 of zeros) and, hence, none of above two inequalities is facet inducing.

We now show that (ii) implies (i). In view of proposition 1.3, it is enough to show that $CW^r_{4,2r+1}(r+1,r+1,r,r,-1,..,-1)$ and $CW^r_{3,r+3}(r+2,r+1,r+1,-1,..,-1)$ are facet inducing. We write the proof of facetness for the first inequality (denoted as v), the proof for the second one is along the same lines and thus omitted. Let $a.x\leq 0$ be a valid inequality over C_n such that $a.x=0$ whenever $v.x=0$ for all $x=\delta(S)$. We prove in claims 1.5-1.7 the existence of a scalar α such that $a=\alpha v$, i.e. $a_{12}=\alpha(r+1)(r/2+1)$, $a_{34}=\alpha r(r-1)/2$, $a_{14}=a_{23}=\alpha r(r+1)/2$, $a_{13}=a_{24}=\alpha r(r+1)$, $a_{1i}=a_{2i}=-\alpha(r+1)$, $a_{3i}=a_{4i}=-\alpha r$, $a_{ij}=\alpha$ for $i,j\in[5,n]$.

Claim 1.5. $a_{ij}=\alpha$ for $i,j\in[5,n]$.

Proof. Take i,j,k in [5,n] and a subset S of [5,n]-{i,j,k} of size r-1; then the sets $\{2,3,j\}\cup I$, $\{2,3,k\}\cup I$, $\{2,3,i,j\}\cup I$, $\{2,3,i,k\}\cup I$ define roots, which, from lemma 1.4, implies that $a_{ij}=a_{ik}$.■

Claim 1.6. $a_{1i}=a_{2i}=-\alpha(r+1)$, $a_{3i}=a_{4i}=-\alpha r$ for $i\in[5,n]$.

Proof. Take $i\in[5,n]$, a subset I of [5,n]-{i} of size r; both sets {2,3,i}∪I and {2,3}∪I define roots, implying: $0 = a.\delta(\{2,3,i\}\cup I) - a.\delta(\{2,3\}\cup I)$ and thus:

(*) $0 = a_{1i} + a_{4i} - a_{2i} - a_{3i}$

Relations: $0 = a.\delta(\{1,i\}) - a.\delta(\{1\})$ and $0 = a.\delta(\{2,i\}) - a.\delta(\{2\})$ yield respectively: $0 = a_{2i} + a_{3i} + a_{4i} - a_{1i} + 2\alpha r$, and $0 = a_{1i} + a_{3i} + a_{4i} - a_{2i} + 2\alpha r$; hence, by substraction, we have: $a_{1i}=a_{2i}$ and $a_{3i}+a_{4i}= -2\alpha r$. This fact, together with (*), implies that $a_{3i}=a_{4i}= -\alpha r$. Finally, given a subset J of [5,n]-{i} of size r-1, we deduce from relation: $0 = a.\delta(\{3,4,i\}\cup I) - a.\delta(\{3,4\}\cup I)$ that $a_{1i}=a_{2i}= -\alpha(r+1)$. ■

<u>Claim 1.7</u>. $a_{12} = \alpha(r+1)(r/2+1)$, $a_{34} = \alpha r(r-1)/2$, $a_{14} = a_{23} = \alpha r(r+1)/2$, $a_{13} = a_{24} = \alpha r(r+1)$.

Proof. Every cut $\delta(\{i\})$ for i=1,2,3,4, is a root; hence, $0 = a.\delta(\{i\})$, yielding respectively:

(R1) $a_{12} + a_{13} + a_{14} = \alpha(r+1)(2r+1)$
(R2) $a_{12} + a_{23} + a_{24} = \alpha(r+1)(2r+1)$
(R3) $a_{13} + a_{23} + a_{34} = \alpha r(2r+1)$
(R4) $a_{14} + a_{24} + a_{34} = \alpha r(2r+1)$

Computing: (R1)-(R2)-(R3)+(R4) gives: $a_{14}=a_{23}$ and $a_{13}=a_{24}$.
Given a subset I of [5,n] of size r, we have: $0 = a.\delta(\{2,3\}\cup I)$, implying: $a_{12} + 2a_{13} + a_{34} = \alpha(3r^2+3r+1)$ and, therefore, using above relations (R2),(R3), we deduce that: $a_{23} = \alpha r(r+1)/2$.
Next, $0 = a.\delta(\{3,4\}\cup I)$, yielding: $a_{13} + a_{23} + a_{14} + a_{24} = 3\alpha r(r+1)$ and thus: $a_{13}=\alpha r(r+1)$.
Finally, we deduce, using (R1), that $a_{12}=\alpha(r+1)(r+2)/2$ and, using (R3), that $a_{34}=\alpha r(r-1)/2$. ■

Next we give a general lifting method which permits to construct new facets from known facets obtained through theorem 1.2. It is in some sense a companion to the lifting method presented in ([DL1], proposition 2.7).

<u>Theorem 1.8</u>. Given integers $p,q=n-p,b_1,..,b_p\geq r\geq 1$, $b_{p+1},..,b_n<0$ with $\Sigma_{1\leq i\leq n}\, b_i = 2r+1$, assume that:
(i) $CW^r_{p,q}(b_1,..,b_n)$ is facet inducing over C_n
(ii) there exist n subsets $T_1,..,T_n$ of [1,n] such that:
  (iia) $T_j\cap[1,p]$ is a (circular) interval of [1,p]
  (iib) $b(T_j)= r+2$
  (iic) the incidence vectors of the sets $T_1,..,T_n$ are linearly independent
Then $CW^{r-1}_{p,q+1}(b_1,..,b_p,-1,..,-1,-2)$ is facet inducing over C_{n+1}.

As example of application of theorem 1.8, we obtain the following new class of facets.

<u>Corollary 1.9</u>. Given integers $b_1,..,b_p\geq r\geq s\geq 0$, $p\geq 5$ and $q = \Sigma_{1\leq i\leq p}\, b_i - 2r - 1 + s$, $n=p+q$, $CW^{r-s}_{p,q}(b_1,..,b_p,-1,..,-1,-2,...,-2)$, where there are q-s components equal to -1 and s components equal to -2, is facet inducing over C_n.

Proof of theorem 1.8. Set $v = CW^r_{p,q}(b_1,..,b_n)$, $v' = CW^{r-1}_{p,q+1}(b_1,..,b_n,-2)$ defined on nodes [1,n+1]. For a subset S of [1,n], we distinguish its cut vector $\delta_n(S)$ in C_n and its cut vector $\delta_{n+1}(S)$ in C_{n+1}. From assumption (i), we can find a family $(\delta_n(S_a): a\in A)$ of $\binom{n}{2}-1$ linearly independent roots of v; w.l.o.g. we can suppose that $b(S_a)=r$ for all $a\in A$, hence $R_1=\{\delta_{n+1}(S_a): a\in A\}$ is a family of $\binom{n}{2}-1$ linearly independent roots of v'. From assumptions (iia),(iib), $R_2 = \{\delta_{n+1}(T_j\cup\{n+1\}): j\in[1,n]\}$ is a family of n roots of v'. We verify that the family $R_1\cup R_2$ is linearly independent. For this, take scalars λ_a, μ_j such that:

(R0) $0 = \Sigma_{a\in A}\,\lambda_a\delta_{n+1}(S_a) + \Sigma_{1\leq j\leq n}\,\mu_j\delta_{n+1}(T_j)$

It suffices to show that $\mu_j=0$ for all j. By taking the projection of (R0) on the coordinate sets $\{(i,j): 1\leq i<j\leq n\}$ and $\{(i,n+1): 1\leq i\leq n\}$, we deduce respectively:

(R1) $0 = \Sigma_{a\in A}\, \lambda_a \delta_n(S_a) + \Sigma_{1\le j\le n}\, \mu_j \delta_n(T_j)$
(R2) $0 = \Sigma_{a\in A}\, \lambda_a X(S_a) + \Sigma_{1\le j\le n}\, \mu_j (1-X(T_j))$
where $X(S)$ denotes the incidence vector of the set S.
By computing the scalar product of vector $b=(b_1,..,b_n)$ with both sides of relation (R2) and using the fact that $b(S_a)=r$ and $b(T_j)=r+2$, we deduce that:
(R3) $r(\Sigma_{a\in A}\, \lambda_a) - (r+1)(\Sigma_{1\le j\le n}\, \mu_j) = 0$
By computing the scalar product of vector v with both sides of relation (R1) and using the fact that $v.\delta_n(S_a)=0$ and $v.\delta_n(T_j) = b(T_j)(2r+1-b(T_j) - r(r+1) = -2$, we deduce that: $\Sigma_{1\le j\le n}\, \mu_j = 0$ and, thus, from (R3), $\Sigma_{a\in A}\, \lambda_a = 0$.
Specifying node 1, we define the sets $B = \{a \in A: 1 \in S_a\}$ and $J = \{j \in [1,n]: 1 \in T_j\}$. Looking at the value taken at coordinate 1 in both sides of (R2) yields:
(R4) $\Sigma_{a\in B}\, \lambda_a - \Sigma_{j\in J}\, \mu_j = 0$
By computing the projection of (R1) on the coordinates $\{(1,i): 2\le i\le n\}$, we obtain:
$0 = \Sigma_{a\in B}\, \lambda_a(1-X(S_a)) + \Sigma_{a\in A-B}\, \lambda_a X(S_a) + \Sigma_{j\in J}\, \mu_j(1-X(T_j)) + \Sigma_{j\in [1,n]-J}\, \mu_j X(T_j)$
and, taking scalar product with vector b in above relation, we deduce that:
$0 = (1-2r)(\Sigma_{a\in B}\, \lambda_a) - (2r+3)(\Sigma_{j\in J}\, \mu_j)$. This fact, together with (R4) implies that: $\Sigma_{j\in J}\, \mu_j = 0$, i.e. the coordinate indexed by 1 of vector $\Sigma_{1\le j\le n}\, \mu_j X(T_j)$ is equal to zero. Of course, we could do the same reasoning with any coordinate i of [1,n] instead of 1 and, therefore, we have that: $0 = \Sigma_{1\le j\le n}\, \mu_j X(T_j)$. Using assumption (iic), we deduce that $\mu_j=0$ for $j=1,..,n$. ∎

Proof of corollary 1.9. In view of proposition 1.3, it suffices to prove that $v_s = CW^{r-s}_{p,q}(r,..,r, -1,..,-1,-2,..,-2)$ (with p times r, $q' = q-s$ times -1 and s times -2) is facet inducing for $p\ge 5$, $pr-q' = 2r+1$, $r\ge s\ge 0$. We prove this result by induction on $s\ge 0$. For $s=0$, v_s is indeed facet inducing. Given $s\in[0,r-1]$, assume that v_s is facet inducing; we show that v_{s+1} is facet inducing with the help of theorem 1.8. Set $b=(r,..,r,-1,..,-1,-2,..,-2)$ (with p times 1, q' times -1 and s times -2) and denote by P, Q′, Q″, respectively, the set of nodes i with $b_i = r, -1, -2$, respectively. Let F denote the family of the incidence vectors $X(S)$ of the subsets S of $P\cup Q'\cup Q''$ such that $S\cap P$ is an interval and $b(S)=r+2$. We show below that the family F has full rank $n\ (= p+q = p+q'+s)$. For this, take scalars $\lambda=(\lambda_1,..,\lambda_n)$ such that $\lambda.X(S)=0$ for all $X(S)$ in F; we prove that $\lambda=0$. For this, we distinguish two cases according to the parity of s.
Assume first that s is even, $s\ge 0$. We describe some members $X(S)$ of F:
(a1) $S=S_1\cup S_2\cup S_3$, S_1 interval of P of size 4, S_2 subset of Q′ of size 3r-2, S_3 subset of Q″ of size s/2
(a2) $S=S_1\cup S_2$, S_1 interval of P of size 2, S_2 subset of Q′ of size r+s-2
(a3) $S=S_1\cup S_2\cup S_3$, S_1 interval of P of size 2, S_2 subset of Q′ of size r+s-4, S_3 subset of Q″ of size 1, if $s\ge 2$
Applying relation: $0 = \lambda.X(S)$ to the members of F of type (a1), we deduce easily that: $\lambda_i=\alpha$ for $i\in P$, $\lambda_i=\beta$ for $i\in Q'$, $\lambda_i=\gamma$ for $i\in Q''$ and thus:
(R1) $4\alpha + \beta(3r-2) + \gamma s/2 = 0$
and, similarly, using members of F of type (a2),(a3):
(R2) $2\alpha + \beta(r+s-2) = 0$
(R3) $2\alpha + \beta(r+s-4) + \gamma = 0$
If $s=0$, we deduce from (R1),(R2) that $\alpha=\beta=0$. If $s\ge 2$, we deduce from (R2),(R3) that $\gamma=2\beta$ and, then, from (R1),(R2) that $\beta(r-s+2)=0$, i.e. $\beta=0$ and thus $\alpha=\gamma=0$.
We now turn to the case when s is odd. Some members $X(S)$ of F are described below:
(b1) $S=S_1\cup S_2\cup S_3$, S_1 interval of P of size 4, S_2 subset of Q′ of size 3r-1, S_3 subset of Q″ of size (s-1)/2
(b2) $S=S_1\cup S_2\cup S_3$, S_1 interval of P of size 4, S_2 subset of Q′ of size 3r-3, S_3 subset of Q″ of size (s+1)/2
(b3) $S=S_1\cup S_2$, S_1 interval of P of size 2, S_2 subset of Q′ of size r+s-2
As before, using above members of F, we deduce that $\lambda_i=\alpha, \beta, \gamma$, for $i\in P, Q', Q''$, respectively and:

$4\alpha + \beta(4r-1) + \gamma(s-1)/2 = 0$
$4\alpha + \beta(3r-3) + \gamma(s+1)/2 = 0$
$2\alpha + \beta r+s-2) = 0$
yielding easily that $\alpha=\beta=\gamma=0$. ■

We conclude this section by giving some more detailed information about inequality $CW^r_{3,n-3}(n-4,r+1,r+1,-1,..,-1)$; it is facet inducing for $(r=0,n\geq 3)$ or $(r\geq 1,n\geq r+6)$. Its number of roots is: $\binom{n-2}{r+2}$ + 2n - 4 ; hence it is a simplicial facet for $(r=0,n\geq 3)$ or for $(r\geq 1,n=r+6)$, actually, for r=0, it is the only class of simplicial CW facets that we know. A second interesting fact is that the order of magnitude of the number of roots is in $O(n^{r+2})$, i.e. polynomial in n of arbitrary degree. The case r=1 in theorem 1.2 is of special interest; the three possible CW facets $CW^1_{3,n-3}(n-4,2,2,-1,..,-1)$, $CW^1_{4,n-4}(n-5,2,1,1,-1,..,-1)$ and $CW^1_{5,n-5}(n-6,1,1,1,1,-1,..,-1)$ for $n\geq 7$ have the same number of roots, namely: $\binom{n-2}{3}$ + 2n - 4 and so all three are simplicial for n=7. It turns out that all CW facets with $r\geq 1$ of C_7 are the above three facets.

2. THE PARACHUTE INEQUALITY: EQUALITY CASE

The **parachute inequality**, denoted as Par_{2k+1}, is defined on the 2k+1 nodes $\{0,1,2,..,k,1',2',..,k'\}$ by:

$$(2.1)\quad Par_{2k+1}.x = \sum_{(i,j)\in P} x_{ij} - \sum_{1\leq i\leq k-1} (x_{0i} + x_{0i'} + x_{ki'} + x_{k'i}) - x_{kk'} \leq 0$$

where P is the path (k,k-1,..,2,1,1',2',..,(k-1)',k'); it was introduced in [DL1] and proved there to be facet inducing for all k odd. Here we show how the number of roots can be expressed in terms of the Fibonacci numbers. Also, we point out the close connection between the parachute inequality and the class of CW facets $CW^r_{2r+3,2}(1,..,1,-1,-1)$ (with r=k-2) ; namely, both admit as common collapsing the following inequality defined on the 2k nodes $\{0,0',1,1',2,2',..,(k-1),(k-1)'\}$:

$$(2.2)\quad Fib_{2k}.x = \sum_{(i,j)\in Q} x_{ij} - \sum_{1\leq i\leq k-1} (x_{0i}+x_{0i'}) - \sum_{1\leq i\leq k-2} (x_{0i}+x_{0i'}) \leq 0$$

where Q is the path (k-1,k-2,..,2,1,1',2',..,(k-2)',(k-1)'). We call above inequality (2.2) the **Fibonacci inequality.**

The well-known Fibonacci sequence is the sequence $(f_1,f_2,..)$ defined recursively by:
$f_1=f_2=1$, $f_{i+2} = f_i + f_{i+1}$ for $i\geq 1$.
Given a path A=(1,2,..,n), a subset S of [1,n] is called **alternated** along path A if $|S\cap\{i,i+1\}|\leq 1$ for all i=1,..,n-1, and **pseudo-alternated** along path A if $|S\cap\{i,i+1\}|=1$ for all i=1,..,j-1,j+1,.., n-1 and $|S\cap\{j,j+1\}|=0$ or 2 for some $j\in[1,n-1]$.
One observes easily that, for n even, the number of pseudo-alternated subsets S along path A=(1,..,n) for which nodes 1,n belong to S is equal to n-1; an easy induction on n shows the following result:

Lemma 2.3. The number of alternated subsets of [1,n] along path (1,2,..,n) is equal to the Fibonacci number f_{n+2}.

Call a cut $\delta(S)$ **symmetric** if, for i=1,2,..,k, i belongs to S if and only if i′ belongs to S, i.e. the involution $\alpha = \prod_{1\le i\le k} (ii')$ belonging to the symmetric group Sym(2k+1) leaves S invariant. We describe below the roots of the parachute inequality.

Proposition 2.4. The roots of the parachute inequality Par_{2k+1} are the cuts $\delta(S)$ for which S is a subset of $[1,k]\cup[1',k']$ of one of the following four types:
Type 1: nodes k,k′ belong to S and S is pseudo-alternated along path P
Type 2: nodes k,k′ do not belong to S and S is alternated along path Q
Type 3: for k odd, node k belongs to S, node k′ does not belong to S and (a) or (b) holds:
(a) $S = \{2',4',...,(k-1)',k\}\cup T$ where T is a subset of {1,2 ,..,k-2} alternated along path (1,2,..,k-2)
(b) $S = \{k,1',(k-1)'\}\cup T\cup V$ where T is a subset of {2,3,..,k-2} alternated along path (2,3,..,k-2) and V is a subset of {2′,3′,..,(k-2)′} such that $V\cup\{1',(k-1)'\}$ is pseudo-alternated along path (1′,2′,..,(k-1)′)
Type 3′: similar to type 3, exchanging nodes i,i′ for all i=1,2,..,k.

There are 2k-1 roots of type 1, all of them linearly independent and the only symmetric root among them is $\delta(\{1,3,..,k,1',3',..,k'\})$ for k odd and $\delta(\{2,4,..,k,2',4',...,k'\})$ for k even. There are f_{2k} roots of type 2, their rank is: $\binom{2k-1}{2} - 2k + 3$ and there are f_k symmetric roots among them. The roots of type 3,3′ exist only for k odd; there are altogether $2(f_k+(k-1)f_{k-1})$ such roots and there are no symmetric roots among them.

Proposition 2.5. (i) the number of roots (including zero, i.e. cut $\delta(\varnothing)$) of the parachute inequality Par_{2k+1} is equal to: $f_{2k} + 2kf_{k-1} + 2f_{k-2} + 2k - 1$ for k odd and $f_{2k} + 2k - 1$ for k even, while the number of (non zero) symmetric roots is always the Fibonacci number f_k.
(ii) The parachute inequality Par_{2k+1} is facet inducing for k odd; for k even, it has dimension $\binom{2k-1}{2} + 2$, but it is not valid.

Proof of propositions 2.4 and 2.5. Given a subset S of $[1,k]\cup[1',k']$, we set: $s=|S\cap[1,k-1]|$ and $s'=|S\cap[1',(k-1)']|$. In order to characterize which cuts $\delta(S)$ are roots of the parachute inequality Par_{2k+1}, we distinguish four cases:

Case 1: $k,k'\in S$
Then $\delta(S)$ is root of Par_{2k+1} if and only if $|\delta(S)\cap P|=2k-2$, i.e. all edges of P but one are edges of $\delta(S)$, i.e. S is pseudo-alternated along path P. So there 2k-1 such roots, among them only one symmetric root: $\delta(\{k,..,3,1,1',3',..,k'\})$ for k odd and $\delta(\{k,..,4,2,2',4',..,k'\})$ for k even.

Case 2: $k,k'\notin S$
Then $\delta(S)$ is root of Par_{2k+1} if and only if $|\delta(S)\cap P|=2(s+s')=2|S|$, i.e. S is alternated along the path (k-1,..,1,1′,..,(k-1)′) on 2k-2 nodes, so, from lemma 2.3, there are f_{2k} such roots. Among them, the number of symmetric roots (including zero) is equal to the number of alternated subsets along path (2,3,..,k-1), i.e. to f_k.

Case 3: $k\in S$, $k'\notin S$
Then $\delta(S)$ is root of Par_{2k+1} if and only if $|\delta(S)\cap P|=k+2s$. Since $|\delta(S)\cap P| = |\delta(S)\cap Path(k,..,1,1')| + |\delta(S)\cap Path(1',..,k')|$, with the first term being less or equal to 2s+2, we have to distinguish two cases:
Case 3a: $|\delta(S)\cap Path(1',..,k')|=k-1$
If k is even, then, necessarily $1'\in S$, contradicting the fact that $|\delta(S)\cap Path(1,1',..,k')|=2s+1$.
If k is odd, then $S\cap\{1',2',..,k'\}=\{2',4',..,(k-1)'\}$ and $|\delta(S)\cap Path(1',1,2,..,k)|=2s+1$, i.e. S is alternated along path (1,2,..,k-2); so there are f_k such roots.
Case 3b: $|\delta(S)\cap Path(1',..,k')|=k-2$

If k is even, then, necessarily, $1' \notin S$, contradicting the fact that $|\delta(S) \cap Path(1',1,2,..,k)| = 2s+2$. If k is odd, then, necessarily, $1',(k-1)' \in S$ and, since $|\delta(S) \cap Path(1',1,..,k)| = 2s+2$, S is alternated along path (2,3,..,k-2) while S is pseudo-alternated along path (1',..,k'); so there are $(k-1)f_{k-1}$ such roots.

Case 4: identical to case 3, exchanging nodes i,i' for i=1,..,k.

Hence, the total number of roots is: $2k-1 + f_{2k} + 2f_k + 2(k-1)f_{k-1} = 2k-1 + f_{2k} + 2kf_{k-1} + 2f_{k-2}$ for k odd and $2k-1 + f_{2k}$ for k even, while the number of non zero symmetric roots is f_k, stating proposition 2.5 (i).

We now prove proposition 2.5 (ii). It was proven in [DL1] that Par_{2k+1} is facet inducing for k odd and that it is not valid for k even. We consider now Par_{2k+1} for k even; then, the set of roots is $R_1 \cup R_2$, where R_i denotes the set of roots of type i, for i=1,2. In order to facilitate the computation of the rank of the set of roots, we use the following notion of **intersection vector**: for a subset S of $[1,k] \cup [1',k']$, define the vector $\pi(S)$ of $\{0,1\}^{k(2k+1)}$ by: $\pi(S)_{ij}=1$ if $i,j \in S$ and $\pi(S)_{ij}=0$ otherwise for all i,j (non necessarily distinct) in $[1,k] \cup [1',k']$. Given a family of subsets $(S_a: a \in A)$ of $[1,k] \cup [1',k']$, the family of cut vectors $(\delta(S_a): a \in A)$ is linearly independent if and only if the family of intersection vectors $(\pi(S_a): a \in A)$ is linearly independent ([DL1]).
First, we check that all roots in $R_1 := \{\delta(S_a): a \in A\}$ are linearly independent. For this, we take a linear combination of their intersection vectors: $\Sigma_{a \in A} \lambda_a \pi(S_a) = 0$. In order to verify that $\lambda_a = 0$ for all a, observe that, for each root $\delta(S_a)$ of R_1, one can find a pair (i,j) such that $\{i,j\} \subseteq S_a$ while $\{i,j\} \not\subseteq S_b$ for the other roots $\delta(S_b)$ of R_1 (for instance, take pair (k-1,k) for root $\delta(\{k,k-1,k-3,....,2,1',3',..,k'\})$).
Next, we check that the rank of the family R_2 is: $\binom{2k-1}{2} - 2k+3$. For this, observe first that the subfamily R_2' of R_2 consisting of all possible singletons and pairs of $[1,k-1] \cup [1',(k-1)']$ has full rank equal to: $2k-2 + \binom{2k-2}{2} - (2k-3) = \binom{2k-1}{2} - (2k-3)$ (easy if one considers the intersection vectors). Then, note that, for every cut $\delta(S)$ of R_2, nodes k,k' do not belong to S and S is alternated along Q, implying that: $x_{kk} = x_{k'k'} = x_{kk'} = x_{ki} = x_{k'i} = x_{i,i+1} = 0$ for $i \in [1,k-1] \cup [1',(k-1)']$ where $x = \pi(S)$ for $\delta(S) \in R_2$. Therefore, we deduce that the rank of R_2 is less or equal to: $\binom{2k+1}{2} - (6k-4) = \binom{2k-1}{2} - (2k-3)$.
Finally, we verify that the family $R_1 \cup R_2'$ is linearly independent, thus stating that the rank of face Par_{2k+1} for k even is: $2k-1 + \binom{2k-1}{2} - (2k-3) = \binom{2k-1}{2} + 2$. Again, we take a linear combination of the intersection vectors: $\Sigma \lambda_a \pi(S_a) + \Sigma \mu_c \pi(T_c) = 0$ where the first sum is over the intersection vectors corresponding to cuts in R_1 and the second one corresponds to cuts in R_2'. It is enough to show that $\lambda_a = 0$ for all a. For this, for the roots $\delta(S_a)$ of R_1 having $\{i,i+1\} \subseteq S_a$ for some i, by looking at the coordinate (i,i+1) in the above linear combination, we obviously deduce that $\lambda_a = 0$. For remaining roots $\delta(S_b)$ of R_1, looking at coordinate (k,i) with $i \in S_b$ yields: $\lambda_b = 0$ too. ■

The Fibonacci inequality Fib_{2k} can be obtained by collapsing nodes k,k' in Par_{2k+1} and calling this new node 0'. It turns out that Fib_{2k} also coincides (up to permutation of the nodes) with the inequality obtained by collapsing nodes 1,n (i.e. one "positive" node and one "negative" node) in $CW^{k-2}_{2k-1,2}(1,..,1,-1,-1)$. Its roots are the cuts $\delta(S-\{k,k'\}+\{0'\})$ for S of type 1 and $\delta(S)$ for S of type 2. So, Fib_{2k} has $2k-1+ f_{2k}$ roots and defines a face of rank $\binom{2k-1}{2} + 2 = \binom{2k}{2} - 2k + 3$. Hence, for k even, Fib_{2k} and Par_{2k+1} have the same rank.
We conclude the section by mentioning the following interesting feature of the parachute facet Par_{2k+1} for odd k; it can be decomposed as linear combination of triangle inequalities:

$$Par_{2k+1}.x = \sum_{1 \le i \le k-1} (T(a_i,i,i+1) + T(a_i,i',(i+1)')) + T(0,1,1') - T(0,k,k')$$

where $a_i=k$ for i odd and $a_i=a_{i'}=0$ for i even, and $T(a,b,c)=x_{bc}-x_{ab}-x_{ac}$ denotes the left hand side of the triangle inequality on nodes a,b,c. A nice property of inequalities $v.x\leq 0$ which can be 'triangulated' is that $v.\delta(S)$ is even for all cuts $\delta(S)$. Similarly, for k even, we have:

$$Fib_{2k}.x = T(0,1,1') + \sum_{1\leq i\leq (k-2)/2} (T(0,2i,2i+1) + T(0,(2i)',(2i+1)') + T(0',2i-1,2i) + T(0',(2i-1)',(2i)'))$$

It is easy to get similar triangulation for Par_{2k+1}, k even and Fib_{2k}, k odd.

3. CHARACTERIZATION OF ALL SWITCHINGS OF SOME FACETS

Switching and permutation are two known operations on facets of the cut polytope. However, in general, the explicit enumeration of all distinct switchings and permutations of a given facet looks a difficult problem. We give below some methods which permit this enumeration for some examples of facets considered before. We refer to [DDL] for the description of all switchings by roots of the facets of the cut cone C_7.

Let F be a facet of the cut polytope P_n supported by inequality: $v.x\leq v_0$. Given a permutation σ of the symmetric group Sym(n) , set $v^\sigma=(v_{\sigma(i)\sigma(j)})_{1\leq i<j\leq n}$, then the inequality $v^\sigma.x\leq v_0$ induces a facet of P_n denoted by $\sigma(F)$. Given a cut $\delta(S)$, define the vector v^S with coordinates: $v^S_{ij}= -v_{ij}$ for $(i,j)\in\delta(S)$ and $v^S_{ij}=v_{ij}$ otherwise, then the inequality: $v^S.x\leq v_0-v.\delta(S)$ induces a facet of P_n, denoted by $r_{\delta(S)}(F)$ and called **switching** of F by $\delta(S)$; if $\delta(S)$ is a root of F, then $r_{\delta(S)}(F)$ is a **switching by root** of F. Note that permutation and switching by roots do not change the right hand side of the inequality supporting F. Two facets F,F' are called permutation (resp. switching) **equivalent** if there exists a permutation σ (resp. a cut $\delta(S)$) such that $F'=\sigma(F)$ (resp. $F'=r_{\delta(S)}(F)$). It may happen that $r_{\delta(S)}(F)=\sigma(F)$ for some cut $\delta(S)$ and some permutation σ; so we are interested in the estimation of the following numbers: number $\nu(F)$ (resp. $\nu_0(F)$) of the switchings (resp. switchings by roots) of F which are pairwise not permutation equivalent.
The **automorphism group** Aut(F) of a facet F of P_n is the subgroup of Sym(n) of all permutations σ such that $\sigma(F)=F$, i.e. σ preserves the set of roots of F, or $v^\sigma=v$ if F is supported by inequality $v.x\leq v_0$. The following easy result can be used as a tool for distinguishing non permutation equivalent facets.

Lemma 3.1. The automorphism groups of permutation equivalent facets are conjugated.

However, in general, switching, even by root, affects the order of the automorphism group. The automorphism group of any facet F acts naturally on the set X of all cuts and, in particular, on the set of roots R(F) of F; denote by $w(F)$ (resp. $w_0(F)$) the number of orbits of X (resp. R(F)) under the action of Aut(F). The following upper bounds hold (see [DL2], [DGriL]):

(3.2) $\nu(F)\leq w(F)$
(3.3) $\nu_0(F)\leq w_0(F)$

We will see below cases when equality is attained (or not) in these bounds. We now describe several examples of facets for which we could find the exact number of non permutation equivalent switchings.

Example 1. F is the hypermetric facet $CW^0_{p,q}(1,..,1,-1,..,-1)$ $(p=q+1)$

Then, the automorphism group of F is Sym(q+1)xSym(q). Set b=(1,..,1,-1,..,-1) with q+1 components 1 and q components -1 and $X_s=\{\delta(S): b(S)=s\}$; then the sets $X_1,X_2,..,X_{q+1}$ form a partition of the set X of all cuts. For any cut $\delta(S)$ of X_s with s=1,..,q+1, facet $r_{\delta(S)}(F)$ is supported by inequality: $Q(b').x \le s(s-1)$ where $b'=(b'_1,..,b'_n)$ consists of q+1-s components 1 and q+s components -1 and its automorphism group is Sym(q+1-s)xSym(q+s). Note that, for s=q+1, $r_{\delta(S)}(F)$ is the well-known facet supporting the **equicut** polytope ([CRS],[DFL]). Therefore, there are exactly $\nu(F)=q+1$ non permutation equivalent switchings of F, among them only $\nu_0(F)=1$ switching by root (F itself). However, the number of orbits of R(F) under action of Aut(F) is: $w_0(F)=q+1$ (i.e. number of distinct values for the pairs $(|S\cap[1,q+1]|, |S\cap[q+2,n]|)$ for roots $\delta(S)$), while the number of orbits of the set of all cuts is: $w(F)=(q+1)(q+2)/2$ (i.e. number of distinct values for above pairs over all cuts)).

Example 2. F is facet $CW^0_{3,n-3}(n-4,1,1,-1,..,-1)$ for $n\ge4$
Set b=(n-4,1,1,-1,..,-1); for any subset S of [2,n], the quantity b(S) takes n distinct values: 2,1,0, -1,-2,...,-(n-3). Denote by X_a the family of cuts $\delta(S)$ with $S\subseteq[2,n]$ and b(S)=a for $a\in A=\{2,1,0, -1,-2,..,-(n-3)\}$. So, the families X_a for $a\in A$ partition the set of all cuts into n classes. Note that X_a is full dimensional for $a\in A-\{1,2,-(n-2),-(n-3)\}$. Also, X_a has cardinality $\binom{n-1}{2-a}$ for $a\in A$. One checks easily that the switchings of F by cuts $\delta(S),\delta(T)$ are permutation equivalent if and only if both cuts belong to the same family X_a; hence, $\nu(F)=n$. These n switchings are the following facets:
F_2 supported by inequality: $Q(n-4,-1,-1,-1,..,-1).x\le2$
F_1 supported by inequality: $Q(n-4,-1,1,-1,..,-1).x\le0$
$F_0=F$
F_{-i} supported by inequality: $Q(n-4,1,1,1..1,-1,..,-1).x\le i(i+1)$ (with i+2 ones in argument of Q) for i=1,2,..,n-3.
Note that the right hand sides of the inequalities supporting the n facets F_a take n-2 distinct values (namely, i(i+1) for i=0,1,2,...,n-3). Also, $Aut(F_a)=Sym(2-a)xSym(n-3+a)$ for $a\in A$. Hence, we have the equalities: $|Aut(F_a)|.|X_a|=(n-1)!$ for all $a\in A$ (see [DGriL] for explanation of this fact). One checks easily that $w(F)=3(n-2)$, $w_0(F)=4$.

Example 3. F is facet $CW^r_{3,n-3}(n-4,r+1,r+1,-1,..,-1)$ for $n\ge r+6$, $r\ge1$ (theorem 1.2)
The automorphism group is Sym(2)xSym(n-3) and the number of orbits of the set of roots is: $w_0(F)=5$ while the total number of orbits is $w(F)=3(n-2)$.

Proposition 3.4. The number of non permutation equivalent switchings (resp. switchings by roots) of facet $F=CW^r_{3,n-3}(n-4,r+1,r+1,-1,..,-1)$ ($n\ge r+6\ge7$) is: $\nu(F)=3(n-2)$ (resp. $\nu_0(F)=5$), i.e. equality holds in bounds (3.2),(3.3).

Proof. The cuts can be written as $\delta(S\cup S')$ with S subset of [2,3], S' subset of [4,n], $s=|S|$, $s'=|S'|$, $0\le s\le2$, $0\le s'\le n-3$. The 3(n-2) orbits of the set X of cuts, as well as the five orbits of the set of roots, are specified by the value of (s,s'). Call $F_{(s,s')}$ the facet obtained by switching F by cut with sizes (s,s') (as defined above). We prove that the facets $F_{(s,s')}$ are pairwise non permutation equivalent. Set $a = (n-4)(r+1) - r(r+1)/2$ (weight of edges (1,2),(1,3) in F) and $b = (r+1)^2 - r(r+1)/2$ (weight of edge (2,3) in F). First, facets $F_{(0,s')}$, $F_{(1,t')}$, $F_{(2,t')}$ with $s',t,t'\in[0,n-3]$, have, respectively, zero, one, two, edges with weight -a; hence, they are pairwise not permutation equivalent. Then, facet $F_{(s,s')}$ contains exactly s' edges with weight n-4; hence, $F_{(s,s')}$, $F_{(s,s'')}$ are not permutation equivalent if $s'\ne s''$, for any s=0,1,2. ■

In the examples below, we only consider the number of distinct switchings by roots.

Example 4. F is facet $CW^r_{p,q}(1,..,1,-1,..,-1)$ ($pr-q=2r+1$, $p\ge5$, $r\ge1$, cf. theorem 1.2)

The automorphism group is D_pxSym(q) where D_p is the dihedral group on p points. The roots are the cuts $\delta(S \cup S')$ with S interval of [1,p], S' subset of [p+1,n] and $|S'|=r(|S|-1)$, $1 \le |S| \le p-2$ ([DL2]); hence the number of orbits of the set of roots is $\omega_0(F) = p-1 = 1+(q+1)/r$.

Theorem 3.5. The number of non permutation equivalent switchings by roots of facet $F = CW^r_{p,q}(1,..,1,-1,..,-1)$ is: $\nu_0(F) = \omega_0(F) = p-1 = 1 + (q+1)/r$.

The basic idea of the proof is that permutation equivalent facets have the same (up to order) **weighted degree sequence**, where, for a facet defining inequality $v.x \le v_0$, the weighted degree of node i is $d(i)=(d_w(i): w \in W)$ with W being the set of possible nonzero values of v_{ij} for $1 \le i < j \le n$ and $d_w(i)$ being the number of edges (i,j) such that $v_{ij}=w$.

Proof of theorem 3.5. Set $v=CW^r_{p,q}(r,..,r,-1,..,-1)$; so $v_{i,i+1}=r(r-1)/2$ for $i \in P=[1,p]$, $v_{ij}=r^2$ for $i,j \in P$, $j>i+2$, $v_{ij}=1$ for $i,j \in Q=[p+1,n]$ and $v_{ij}=-r$ for $i \in P$, $j \in Q$. So, there are 4 (resp. 2) possible nonzero values for v_{ij} if $r \ge 2$ (resp. $r=1$). Denote by F_s the facet obtained by switching F by root $\delta([1,s] \cup [p+1,p+r(s-1)])$ for $0 \le s \le p-2$ (i.e. roots representative of the p-1 orbits of the set of roots of F), $F_0=F$; we show that the facets F_s, $0 \le s \le p-2$, are pairwise not permutation equivalent by checking that they have distinct (up to order) weighted degree sequences. We must distinguish the cases $r=1$, $r \ge 2$.

Case: r=1. Then $p=q+3$. The possible nonzero values of the edgeweights in F or any switching of F are: 1, -1. For facet F, the possible weighted degrees are:

* for node $i \in P$, $d(i)=(d_1(i),d_{-1}(i))=(q,q)$
* for node $i \in Q$, $d(i)=(q-1, q+3)$

So, the weighted degree sequence of F consists of (q,q) repeated p times, (q-1,q+3) repeated q times. Facet F_s is supported by inequality: $Q(b_1',..,b_n').x + \Sigma_{(i,j) \in C(S)} x_{ij} \le 0$, where C(S) is the weighted cycle on (1,2,..,p) with weight 1 on edges (1,p),(1,2),(s-1,s),(s,s+1) and weight -1 on the other edges and $b'=(-1,..,-1,1,..,1,-1,..,-1)$ consisting of s components -1, then q+2 components 1 and last q-s+1 components -1. By direct check, one computes the weighted degree sequence of facet F_s. So, if $s=1$, the possible weighted degrees for F_1 are as follows:

* for nodes 1,2,p, $d(i)=(q+2,q-2)$
* for nodes $i \in [3,p-1]$, $d(i)=(q+1,q-1)$
* for nodes $i \in Q$, $d(i)=(q,q+2)$

Hence, the weighted degree sequence of F_1 consists of (q,q+2) repeated q times, (q+1,q-1) repeated q times and (q+2,q-2) repeated 3 times.
Finally, if $s \ge 2$, the weighted degrees of F_s are as follows:

* for nodes 1,s, $d(i)=(q+1,q-1)$
* for nodes $i \in [2,s-1]$, $d(i)=(q,q)$
* for nodes s+1,p, $d(i)=(q+2,q-2)$
* for nodes $i \in [s+2,p-1]$, $d(i)=(q+1,q-1)$
* for nodes $i \in [p+1,p+s-1]$, $d(i)=(q+1,q+1)$
* for nodes $i \in [p+s,n]$, $d(i)=(q,q+2)$

Hence, the weighted degree sequence of F_s consists of (q,q) repeated s-2 times, (q,q+2) repeated q-s+1 times, (q+1,q-1) repeated q-s+3 times, (q+1,q+1) repeated s-1 times and (q+2,q-2) repeated twice. Obviously, the weighted degree sequences of facets F_s, $0 \le s \le p-1$, are pairwise distinct, implying that $\nu_0(F)=p-1$.

Case: r≥2. The possible nonzero values for the edgeweights in a switching of F are: 1, -1, r, -r, $a=r^2$, -a, $b=r(r-1)/2$, -b. Set $d(i)=(d_1(i),d_{-1}(i),d_r(i),d_{-r}(i),d_a(i),d_{-a}(i),d_b(i),d_{-b}(i))$.
The weighted degrees for facet F are as follows:

* for nodes $i\in P$, $d(i)=(0,0,0,q,p-3,0,2,0)$
* for nodes $i\in Q$, $d(i)=(q-1,0,0,p,0,0,0,0)$

The weighted degrees for facet F_1 are as follows:

* for node $i=1$, $d(i)=(0,0,q,0,0,p-3,0,2)$
* for nodes $i=2,p$, $d(i)=(0,0,0,q,p-3,0,1,1)$
* for nodes $i\in[3,p-1]$, $d(i)=(0,0,0,q,p-3,0,2,0)$
* for nodes $i\in Q$, $d(i)=(q-1,0,1,p-1,0,0,0,0)$

The weighted degrees for facet F_s, $s\geq 2$, are as follows; setting $s'=r(s-1)$:

* for nodes $i=1,s$, $d(i)=(0,0,q-s',s',s-2,p-s-1,1,1)$
* for nodes $i\in[2,s-1]$, $d(i)=(0,0,s',q-s',p-3,p-s,2,0)$
* for nodes $i=s+1,p$, $d(i)=(0,0,s',q-s',p-s-2,s-1,1,1)$
* for nodes $i\in[s+2,p-1]$, $d(i)=(0,0,s',q-s',p-s-3,s,2,0)$
* for nodes $i\in[p+1,p+s']$, $d(i)=(s'-1,q-s',p-s,s,0,0,0,0)$
* for nodes $i\in[p+s'+1,n]$, $d(i)=(q-s'-1,s',s,p-s,0,0,0,0)$

Then, one checks easily that the weighted degree sequences of facets F_s, $0\leq s\leq p-1$, are pairwise distinct. ■

Example 5. F is the facet Gr_7 ([DL1])
The facet Gr_7 is the simplicial facet of C_7 supported by inequality:

$$(3.6)\quad \sum_{1\leq i<j\leq 4} x_{ij} + x_{56} + x_{57} - x_{67} - x_{16} - x_{36} - x_{27} - x_{47} - 2\Big(\sum_{1\leq i\leq 4} x_{5i}\Big) \leq 0$$

The automorphism group of Gr_7 is the subgroup of Sym(7) generated by the three involutions (13), (24) and (14)(23)(67) ((ij) denoting the transposition on i,j), so it is isomorphic to Sym(2)xSym(2)xSym(2). There are seven orbits of the set of roots and it can be checked that $\nu_0(F)=7$, i.e. equality holds in bound (3.3).
One can see by direct check that the seven orbits of R(F) have respective lengths: 1,2,2,4,4,4,4 while the automorphism groups of the facets obtained by switching F by a root of each of these seven orbits have respective orders: 8,4,4,2,2,2,2.

Example 6. F is facet Par_7 (see section 2)
This example presents a lot of beautiful symmetries that we describe in more detail. The automorphism group of Par_7 is the subgroup of Sym(7) generated by the involution $\alpha=(11')(22')(33')$, so it is ismorphic to Sym(2) . Facet Par_7 has 21 roots partitioned into 3 classes: $R_a=\{\delta(a_i)\colon i\in[0,6]\}$, $R_b=\{\delta(b_i)\colon i\in[0,6]\}$ and $R_c=\{\delta(c_i)\colon i\in[0,6]\}$ where a_i for $i=0,1,..,6$ denote respectively the sets: $\varnothing,\{2\},\{2'\},\{1,3,2'\},\{1',3',2\},\{2,1'\},\{2',1\}$, b_i for $i=0,1,..,6$ denote respectively the sets: $\{2,2'\},\{1'\},\{1\},\{2,3,1',3'\},\{2',3',1,3\},\{2,3'\},\{2',3\}$ and c_i for $i=0,1,..,6$ denote respectively the sets: $\{1,3,1',3'\},\{1',3\},\{1,3'\},\{1',3',3\},\{1,3,3'\},\{1,2,3'\},\{1',2',3\}$. Each class R_a,R_b,R_c is the union of 4 orbits (one orbit of size 1 for the symmetric root and three orbits of size 2).
Denote by $F_a=Par_7, F_b, F_c$ the facets obtained by switching Par_7 by the symmetrical roots a_0,b_0,c_0, respectively. The facets F_a,F_b,F_c are not permutation equivalent, however they have the same automorphism group: $\{id,\alpha\}$.
We consider the following involutions: $\pi_1=(03)(13')(1'2')$, $\pi_2=\alpha\pi_1\alpha$, $\pi_3=(02)(1'3')(32')$, $\pi_4=\alpha\pi_3\alpha$, $\pi_5=(01)(21')(2'3')$ and $\pi_6=\alpha\pi_5\alpha$. Then, it turns out that, for $i\in[1,6]$, the facet obtained by switching of Par_7 by root $\delta(a_i)$ (resp. $\delta(b_i)$, $\delta(c_i)$) coincides with the facet obtained by permutation of Par_7 by π_i, yielding that: $\nu_0(Par_7)=3$.

Finally we mention three more curiosities on the roots of Par_7 .
(a) all subsets of $\{1,1',2,2',3,3'\}$ can be generated by taking symmetric differences of members of the set $\{a_i\colon i\in[1,6]\}$, or of $\{b_i\colon i\in[1,6]\}$, or of $\{c_i\colon i\in[1,6]\}$.

(b) c_0 is complement of $b_0\Delta\{0\}$, $c_i=b_i\Delta\{x\}$ with $x=3,3',2,2',1,1'$, for $i=1,2,3,4,5,6$ respectively.
(c) the group generated by the seven involutions α and π_i for $i\in[1,6]$ is the dihedral group D_7 ,the symmetry group of the regular 7-gon; it turns out that the graph supporting Par_7 is the complement of a cycle of length 7 (see [DGriL] for explanation of this fact).

Most of above symmetries are lost for the parachute facet Par_{2k+1} with $k\geq 5$, k odd. The automorphism group of Par_{2k+1} is still the group of order 2 generated by involution $(11')(22')..(kk')$. Counting of the number of switchings is now more difficult; for example, Par_{11} admits two switchings (by roots $\delta(\{2,2'\}),\delta(\{3,3'\})$) which are not permutation equivalent but still have the same weighted degree sequence. The number of orbits of the set of roots of Par_{2k+1} is: $w_0(Par_{2k+1})= 3f_k/2 + f_{2k}/2 + (k-1)f_{k-1} + k$ (number of symmetric roots plus one half of number of non symmetric roots). We conjecture that equality: $\nu_0(Par_{2k+1})=w_0(Par_{2k+1})$ holds for k odd,$k\geq 5$.

Remark 3.7. Total number of facets of the cut cone C_7
For a facet F of C_n, denoting by $F=F_0,F_1,..,F_{a-1}$ $(a=\nu_0(F))$ its non permutation equivalent switchings, the number of distinct facets which are permutation and/or switching equivalent to F is equal to: $\Sigma_{0\leq i\leq \nu_0(F)-1}\ (n!)/(|Aut(F_i)|)$ denoted below as N(F) ([DGriL]).
Using this enumeration formula, above computations of automorphism groups and the fact that C_7 has precisely eleven types (up to switching and permutation) of facets ([G]) (namely Par_7, Gr_7, three CW facets with r=1 and six CW facets with r=0, see also [DDL]), one obtains that C_7 has exactly **38780** facets.

The computation is easy: we indicate for each of the eleven types of facets of C_7 the orders of the automorphism groups of its distinct switchings by roots:

* facet Par_7 has 3 switchings with automorphism groups of orders 2,2,2; so $N(Par_7)=7560$.
* facet Gr_7 has 7 switchings with automorphism groups of orders 8,4,4,2,2,2,2; so $N(Gr_7)=13230$.
* facet $F=CW^0_{2,5}(1,1,-1,0,0,0,0)$ has only one switching and its automorphism group has order 2!4!; so N(F)=105.
* facet $F=CW^0_{3,4}(1,1,1,-1,-1,0,0)$ has only one switching and its automorphism group has order 2!3!2!; so N(F)=210.
* facet $F=CW^0_{4,3}(1,1,1,1,-1,-1,-1)$ has only one switching and its automorphism group has order 3!4!; so N(F)=35.
* facet $F=CW^0_{3,4}(2,1,1,-1,-1,-1,0)$ has 2 switchings whose automorphism groups have orders 2!3! and 4!; so N(F)=630.
* facet $F=CW^0_{3,4}(3,1,1,-1,-1,-1,-1)$ has 2 switchings whose automorphism groups have orders 2!4! and 5!; so N(F)=147.
* facet $F=CW^0_{3,4}(2,2,1,-1,-1,-1,-1)$ has 3 switchings whose automorphism groups have orders 2!4!, 2!3! and 2!5!; so N(F)=546.
* facet $F=CW^1_5(1,1,1,1,1,-1,-1)$ has 4 switchings whose automorphism groups have orders 20, 2!2!, 2!2!, 2!; so N(F)=5292.
* facet $F=CW^1_{3,4}(3,2,2,-1,-1,-1,-1)$ has 5 switchings whose automorphism groups have orders 2!4!, 4!, 3!, 2!2!2!, 2!3!; so N(F)=2205.
* facet $F=CW^1_{4,3}(2,2,1,1,-1,-1,-1)$ has 7 switchings whose automorphism groups have orders 3!2!, 3!, 2!, 2!, 2!, 3!2!, 2!2!; so N(F)=8820.

REFERENCES

[BGM] F.Barahona, M.Grötschel and A.R.Mahjoub, Facets of the bipartite subgraph polytope, Mathematics of Operations Research 10 (1985), 340-358.

[BM] F.Barahona and A.R.Mahjoub, On the cut polytope, Mathematical Programming 36 (1986), 157-173.

[CRS] M.Conforti,M.R.Rao and A.Sassano, The equipartition polytope: parts I&II, IASI-CNR research reports n.194-195, Rome (1987).

[D1] M.Deza, On the Hamming geometry of unitary cubes, Doklady Akad. Nauk. USSR. (in Russian) vol.134 (1960), 1037-1040; English translation in Soviet Physics Dokl. 5 (1961), 940-943.

[D2] M.Deza, Matrices de formes quadratiques non negatives pour des arguments binaires, Comptes Rendus de l'Academie des Sciences de Paris, vol.277 (1973), 873-875.

[DDL] C.De Simone, M.Deza and M.Laurent, Collapsing and lifting for the complete cut cone, IASI-CNR research report n.265, Rome (July 1989); submitted.

[DFL] M.Deza, K.Fukuda and M.Laurent, The inequicut cone, Research report n.89-04, University of Tsukuba, Tokyo, GSSM (September 1989); submitted.

[DGriL]M.Deza, V.P.Grishukhin and M.Laurent, The symmetries of the cut polytope and of some relatives, Research report B-228, Tokyo Institute of Technology (December 1989); submitted.

[DGröL]M.Deza, M.Grötschel and M.Laurent, Clique-Web facets for multicut polytopes, Research report, University of Augsburg (September 1989); submitted.

[DL1] M.Deza and M.Laurent, Facets for the complete cut cone I, Research Memoran dum RMI 88-13, University of Tokyo (December 1988); submitted.

[DL2] M.Deza and M.Laurent, Facets for the complete cut cone II: Clique-Web inequalities, Document du Lamsade n.57, Université Paris Dauphine (1989); submitted.

[G] V.P.Grishukhin, All facets of the Hamming cone for n=7 are known, European Journal of Combinatorics 11 (1990).

The Fastest Algorithm for the Pert Problem with AND-and OR-Nodes (The New-Product-New Technology Problem)

E.A. Dinic
The Institute for System Studies
Moscow, USSR

Extended Abstract

We consider the problem of finding the early set-up periods of new technologies Q_l and of beginning the production of new product P_k. The link from Q_l to P_k means that the technology Q_l puts the product P_k on the market with a delay of $qP_{lk} \geq 0$. The link from P_k to Q_l means that the technology Q_l may be set up not earlier than in a period $pq_{kl} \geq 0$ since first putting the product P_k on the market. (The model assumption is that the quantity of produced items is not taken into account.) In other words, any technology may be set up as soon as all the necessary raw materials are ready for use (the early term - the maximum period of readiness), any product may be considered obtained if at least one technology used for its production puts it on the market (the early term - the minimum period of production).

The mathematical model of the given problem is the hybrid of the PERT (longest route problem) and the shortest route problem. Let's describe the formalization in its general type, and later on we shall give its concrete version for the given problem. Let's consider an oriented network $G = (v, E)$ with nodes $v \in V$ of AND and OR types and with weights t_{ij} on arcs $(v_i, v_j) \in E$. We are to calculate the function $T(v_j)$ on nodes $v_j \in V$ under conditions

$$T(v_j) = \max(0,\ v_i + t_{ij}),\ v_j \quad \text{of AND type},\ (v_i, v_j) \in E,$$
$$T(v_j) = \min(\infty,\ v_i + t_{ij}),\ v_j \quad \text{of OR type},\ (v_i, v_j) \in E.$$

For the New-Product-New-Technology Problem defined above the network is bipartite: one part consists of AND-nodes Q_l and the other - of OR-nodes P_k.

Extreme cases of the hybrid problem are the pure networks, which have nodes of one fixed type. Therefore an algorithm for the general case will not be faster than some PERT algorithm and some algorithm for finding the shortest route in the network. We put forward an algorithm, which is a hybrid of the linear PERT algorithm and some shortest route algorithm of Dijkstra type. It's running time has the summary estimate. The estimate of Dijkstra-algorithm is quadratic from (V). It is possible to use its modification with the address manual, operating either with positive or with integer arc weights. Let $t_{m|n}$ be equal to $\min\{t_{ij}\}$ in the first case and to one of the shortest route - $0(|V|+|E|+\max\{T(v)\}/t_{m|n})$, and the size of the additional storage - $0(\max\{t_{ij}\}/t_{m|n})$. Now let's pass over to the description of the algorithm. At the beginning of the algorithm operation we shall assume that $T(v_j)=0$ for all AND-nodes and $T(v_j)=\infty$ for all OR-nodes $v_j \in V$.

AND-iteration. Find the nonmarked AND-node v_i entered by no nonmarked arc. Mark v_i and declare $T(v_i)$ to be of the true value. Make observation of v_i: for each arc $(v_i, v_j) \in E$ mark this arc and recalculate:

$$T(v_j) = \max(T(v_j),\ T(v_i)+t_{ij}),\ v_i \quad \text{of AND-type, or}$$
$$T(v_j) = \min(T(v_j),\ T(v_i)+t_{ij}),\ v_j \quad \text{of OR-type}$$

After that we find another AND-node of this type and work with it in the same way. Repeat it till each nonmarked AND-node is enterd by at least one nonmarked arc. Afterwards pass over to the OR-iteration.

OR-iteration. Calculate $\min T(v) = T(v_i) \neq \infty$ for all nonmarked nodes of OR-type. Mark v_i, declare $T(v_i)$ to be of the true value and make observation of v_i (just the same as in AND-iteration). After that go to the AND-iteration.

In case there are no nonmarked OR-nodes v_i with $T(v_i) \neq \infty$ assume $T(v_i) = \infty$ for all nonmarked nodes v_j of AND-type and stop the work of the algorithm.

To ensure the correct operation of this algorithm it is enough (but not obligatory) to have in the network G contours, consisting of arcs with zero-weight only.

There is another problem relevant to the discussed formalization. Let's consider the network consisting of logical elements with monotone boolean

functions at their entrances and delays on the links. It is needed to calculate the moments of the elements reaction. Fore such network one can easily build an equivalent network consisting of AND-and OR-elements only (with the help of DNF- or CNF-presentation of the logical functions for example).

PROBABILISTIC ANALYSIS OF THE GENERALISED ASSIGNMENT PROBLEM

Martin Dyer*
School of Computer Studies, University of Leeds, Leeds, U.K.

and

Alan Frieze†
Department of Mathematics, Carnegie-Mellon University,
Pittsburgh, U.S.A.

April 2, 2022

Abstract

We analyse the *generalised assignment problem* under the assumption that all coefficients are drawn uniformly and independently from [0,1]. We show that such problems can be solved exactly with high probability, in a well-defined sense. The results are closely related to earlier work of Lueker, Goldberg and Marchetti-Spaccamela and ourselves.

1 Introduction

We are concerned here with the *generalised assignment problem*:

$$\begin{array}{lllll} \text{Maximise} & z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} & & & \\ \text{subject to} & \sum_{j=1}^{n} a_{ij}x_{ij} & \leq & b_i = \beta_i n & (i = 1,2,\ldots,m) \\ & \sum_{i=1}^{m} x_{ij} & = & 1 & (j = 1,2,\ldots,n) \\ \text{and all} & x_{ij} \in \{0,1\}. & & & \end{array} \tag{1}$$

This problem has applications in machine scheduling and elsewhere. See, for example, [1, 5, 8]. Here we give a *probabilistic analysis* of (1), under the assumption that all a_{ij}, c_{ij} are independently and uniformly distributed in [0,1]. We examine the asymptotic behaviour as $n \to \infty$, while m and the β_i remain fixed. (The assumption that m is constant while $n \to \infty$ is realistic, for example, in the machine-scheduling application.) We assume $m > 1$, since otherwise the problem is obviously trivial.

We have chosen the maximising form for (1). All our results apply equally well to the minimising form, since

*Supported by NATO grant RG0088/89
†Supported by NSF grant CCR-8900112 and NATO grant RG0088/89

the constraints imply that the objective function

$$z = n - \sum_{i=1}^{m}\sum_{j=1}^{n}(1 - c_{ij})x_{ij}.$$

and $(1 - c_{ij})$ has precisely the same distribution as c_{ij}.

The work presented here is closely related to, and in part rests on, our earlier paper [2] on the multidimensional knapsack problem. (This was itself an extension of work of Lueker [6] and Goldberg and Marchetti-Spaccamela [3].) As in that paper, the main technique here is to relate the solution of the integer program (1) to its linear programming (LP) relaxation:

$$\text{(1) with } x_{ij} \in \{0,1\} \text{ replaced by } x_{ij} \geq 0. \tag{2}$$

As in [2], we use strongly the fact that the only conditioning on the data imposed by the solution of (2) arises from satisfying the primal and dual feasibility conditions (which are simply linear inequalities). We make use of well-known ideas from LP theory throughout the paper without comment. We use similarly standard methods from probability theory, in particular certain probability inequalities. See [7] for a useful survey of many such inequalities.

However, although our methods are similar to those of [2], extending a probabilistic analysis to an apparently similar problem is not straightforward, since changed structure produces different conditioning. Nevertheless, we will show that in this case the difficulties can be successfully negotiated. There is one proviso. For technical reasons, we must assume the following condition on the β_i holds. Consider the vector β as a point in R^m. Let the set $\mathcal{F}$ be (the convex set of) those β for which (2) remains feasible with probability tending to 1 as $n \to \infty$. We clearly need $\beta \in \mathcal{F}$ in order that the feasibility of (1) does not dominate the analysis. We assume something stronger, however. Let $\mathcal{F}(\eta)$ be the set of β such that (2) remains feasible (with probability tending to 1) when the β_i are (all) reduced to $\beta_i - \eta$, for given $\eta > 0$. If $\beta \in \mathcal{F}(\eta)$, we will call (1) (or (2)) *η-feasible*. We require η-feasibility for some $\eta > 0$. It is clear that $\operatorname{int}\mathcal{F} = \bigcup_{\eta>0}\mathcal{F}(\eta)$. Thus, in a certain sense, our analysis covers "almost all" β's of interest.

From an algorithmic viewpoint, we do not require conditions for η-feasibility (for some $\eta > 0$). However, various simple sufficient conditions can be derived. For example,

$$\eta = \min_{1\leq i\leq m}\beta_i - \frac{1}{n(m+1)} > 0,$$

is such a condition. To see this, consider the solution to (2) given by setting $x_{ij} = 1$ in column j for the i which gives the minimum a_{ij} in that column. Since this minimum is $1/(m+1)$ in expectation (the minimum of m uniform [0,1]'s), and will occur in each row with probability $1/m$, the sufficiency of the condition follows easily from the law of large numbers.

The results we prove are specifically as follows. The first relates the objective values of (1) and (2). (Natural logarithms are used throughout.)

Theorem 1 *Let (1) be η-feasible for some $\eta > 0$, and z_{IP}, z_{LP} be the optimal values for (1), (2) respectively. Then there exist constants $0 < p(m) < 1$ and $\alpha(m,\eta) > 0$ such that, for large n,*

$$\Pr(z_{LP} - z_{IP} > t\alpha(\log n)^2/n) < p^t, \quad (t = 1, 2, \ldots, \lfloor n/(\log n)^2 \rfloor),$$

$$\text{and } E(z_{LP} - z_{IP}) < (2\alpha/(1-p)^2)(\log n)^2/n.$$

Moreover, if $\mathcal{N}$ is the number of variables whose values differ in the two optimal solutions, then there exists $\sigma(m,\eta)$ such that

$$E(\mathcal{N}) < \sigma \log n$$

□

The second shows that, with any specified probability of error, (1) can be solved in polynomial time.

Theorem 2 *Let (1) be η-feasible for some $\eta > 0$, and $\tau > 0$ a probability. Then there exists a constant $d(m,\eta,\tau)$, such that a proven optimal solution can be obtained by a branch-and-bound search algorithm in time $O(n^d)$ with probability at least $(1-\tau)$.* □

The plan of the paper is as follows. In §2 we give some preliminary results on the linear programming (LP) relaxation of (1). In §3 we prove Theorem 1 and in §4 we prove Theorem 2. Finally §5 gives some brief concluding remarks.

2 Linear Programming Preliminaries

We show here some simple properties of the LP problem (2) and its dual:

$$\begin{array}{lrcll} \text{Minimise} & \sum_{i=1}^m b_i u_i + \sum_{j=1}^n v_j & & & \\ \text{subject to} & a_{ij}u_i + v_j & \geq & c_{ij} & (\forall i,j) \\ & u_i & \geq & 0 & (\forall i). \end{array} \tag{3}$$

Lemma 3 below has been given by Benders and Vannunen [1], with a different proof and a slightly weaker conclusion.

Lemma 3 *Let $\{x_{ij}^B\}$ be any basic feasible solution to the system*

$$\begin{array}{lrcll} \text{subject to} & \sum_{j=1}^n a_{ij}x_{ij} & \leq & b_i & (i = 1, 2, \ldots, m) \\ & \sum_{i=1}^m x_{ij} & = & 1 & (j = 1, 2, \ldots, n) \\ \text{and all} & x_{ij} \geq 0. & & & \end{array} \tag{4}$$

and let $F^B = \{(i,j) : 0 < x_{ij}^B < 1\}, E^B = \{j : \exists (i,j) \in F^B\}$. Then (a) $|E^B| \leq m$, and (b) $|F^B| \leq 2m$.

Proof Any basic feasible solution to (4) is an optimal solution for some objective function

$$\text{Maximise } z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}.$$

In such a solution, for each j, there is an $i^*(j)$ such that $x_{i^*j} > 0$. We use the constraints $\sum_{i=1}^{m} x_{ij} = 1$ to eliminate the x_{i^*j}. This will give an LP:

$$\begin{array}{lll} \text{Maximise} & z = \sum_{j=1}^{n}\sum_{i\neq i^*} c'_{ij}x_{ij} & \\ \text{subject to} & \sum_{j=1}^{n}\sum_{i\neq i^*} a'_{ijk}x_{ij} \leq b'_k & (k=1,2,\ldots,m) \\ & \sum_{i\neq i^*} x_{ij} \leq 1 & (j=1,2,\ldots,n) \\ \text{and all} & x_{ij} \geq 0. & \end{array} \tag{5}$$

Now in the LP (5), all constraints $\sum_{i\neq i^*} x_{ij} \leq 1$ are non-binding at the optimum, and hence redundant. Thus the solution will be the same as that to

$$\begin{array}{lll} \text{Maximise} & z = \sum_{j=1}^{n}\sum_{i\neq i^*} c'_{ij}x_{ij} & \\ \text{subject to} & \sum_{j=1}^{n}\sum_{i\neq i^*} a'_{ijk}x_{ij} \leq b'_k & (k=1,2,\ldots,m) \\ \text{and all} & x_{ij} \geq 0. & \end{array} \tag{6}$$

Let $F = \{(i,j) : x_{ij} > 0 \text{ in the optimal solution to (5)}\}$ and $E = \{j : \exists (i,j) \in F\}$. Clearly $|E| \leq |F|$, and since there is an optimal basic solution, $|F| \leq m$. Now $x_{i^*j} < 1$ if and only if $j \in E$. Thus $E^B = E$, proving (a). Also $F^B = F \cup \{(i^*, j) : j \in E\}$, and since the union is disjoint $|F^B| = |F| + |E| \leq 2m$, proving (b). □

Lemma 4 *Any basic feasible solution to (3) can be completely specified by giving $E^B \subseteq \{1,2,\ldots,n\}$ with $|E^B| \leq m$, and $F^B \subseteq \{(i,j) : j \in E^B\}$ with $|F^B| \leq 2m$.*

Proof Consider the dual (3). We assume, without loss of generality, that it is non-degenerate. If the set F^B is as defined in Lemma 3, then the set of equations

$$a_{ij}u_i + v_j = c_j \qquad (i,j) \in F^B$$

will uniquely determine the non-zero u_i. The remaining v_j can now be determined by $v_j = \max_i\{c_{ij} - a_{ij}u_i\}$. This completely specifies the dual basic feasible solution, and the primal solution can be recovered by complementary slackness. □

The method of specifying a dual basic feasible solution implied by Lemma 4 is central to our arguments below. It has the following important consequence.

Corollary 5 *(Assuming non-degeneracy) there are less than $(6m)^m n^m$ basic feasible solutions to (3).*

Proof There are clearly at most

$$\binom{n}{m}\binom{m^2}{2m} < (en/m)^m (em^2/2m)^{2m} = (e^3 mn/4)^m < (6mn)^m$$

ways of selecting E^B then F^B. □

The non-degeneracy assumption holds with probability 1 in our model. Thus, as $n \to \infty$ with m fixed, we will have $O(n^m)$ (i.e. a polynomial number of) dual basic feasible solutions to consider. This should be contrasted with the m^n (i.e. an exponential number of) potential primal basic feasible solutions.

We now show that the η-feasibility condition discussed in §1 implies a bound on the optimal dual variables in (2). For ease of notation let us write (2) in "matrix form", i.e.

$$LP(b): \quad z_{LP}(b) = \max z$$
$$\text{subject to } z = cx, \quad Ax \leq b, \quad Ex \leq e_n, \quad x \geq 0.$$

where, in particular, e_n denotes an n-vector of 1's.

Lemma 6 *Let $\eta > 0$. If (2) is η-feasible and $u_i (i = 1, \ldots, m)$ are the dual variables in its optimal solution, then*

$$\sum_{i=1}^{n} u_i \leq 1/\eta$$

Proof Let x be any feasible solution to $LP(b - n\eta e_m)$. Clearly $cx \geq 0$. The optimal dual variables for $LP(b)$ satisfy

$$uA + vE \geq c, \quad u \geq 0, \quad ub + ve_n \leq n, \tag{7}$$

the last inequality following from the duality theorem and the obvious upper bound on the primal objective function. Hence

$$\begin{aligned} n &\geq ub + ve_n \\ &\geq u(Ax + \eta n e_m) + ve_n \\ &= uAx + \eta n u e_m + vEx \\ &= (uA + vE)x + \eta n u e_m \\ &\geq cx + \eta n u e_m \\ &\geq \eta n u e_m. \end{aligned}$$

Thus $ue_m \leq 1/\eta$, as required. □

The last result in this section relates the optimal solution of (2) to "nearly optimal" solutions when the right hand sides are changed.

Lemma 7 *Let y' be an optimum basic feasible solution to $LP(b')$ and $x = y' + \theta$ any feasible solution to $LP(b)$. Let $u \geq 0$ be the optimum dual variables for the inequalities in $LP(b')$ and $\bar{c} \geq 0$ the reduced costs associated with the nonbasic variables in y'. Then*

$$cx \leq z_{LP}(b) + \bar{c}\theta + u(b - Ax)$$

Proof From the objective row of the basis associated with y' we have

$$cx = \bar{c}x + u(b' - Ax) + cy'.$$

Let $y \geq 0$ solve $LP(b)$, so

$$z_{LP}(b) = cy = \bar{c}y + u(b' - Ay) + cy'$$

Thus

$$\begin{aligned} cx - cy &= \bar{c}(y' + \theta) - \bar{c}y + u(Ay - Ax) \\ &= \bar{c}\theta - \bar{c}y + u(Ay - Ax) && (\bar{c}y' = 0) \\ &\leq \bar{c}\theta + u(Ay - Ax) && (\bar{c} \geq 0, y \geq 0) \\ &\leq \bar{c}\theta + u(b - Ax) && (Ay \leq b, u \geq 0) \end{aligned}$$

□

3 Proof of Theorem 1

Assume $1 > \eta > 0$ is given, and that (1) is η-feasible. Let $K = \frac{9}{20}m^2\eta \log n$, $k = \lfloor 18m^2 \log n \rfloor$, $M = 2^{m+9}/\eta^2$ and $\epsilon = 80m^3M \log n/n$.

Now, for $i = 1, \ldots, m$, let $b'_i = b_i - K$. Let x^B solve $LP(b')$. From Lemma 6 the dual variables u_i for $LP(b')$ satisfy

$$\sum_{i=1}^{m} u_i \leq \frac{1}{\eta - K/n} < 2/\eta,$$

for all n large enough. So $u_i < 2/\eta$, for all i.

Let E^B, F^B be as defined in §2. For each $j = 1, \ldots, n$ define $i^*(j)$ as follows. For $j \notin E^B$, let $i^*(j)$ be the (unique) i such that $x^B_{ij} = 1$. For $j \in E^B$, choose $i^*(j)$ arbitrarily from $\{i : (i,j) \in F^B\}$. We define a "rounded" zero-one solution ξ^B by

$$\xi^B_{ij} = 1 \text{ if and only if } (i,j) = (i^*, j).$$

Now let

$$\begin{aligned} \Delta^B_i &= b_i - \textstyle\sum_{j=1}^{n} a_{ij}\xi^B_{ij} && (u_i \neq 0) \\ &= K && \text{otherwise} \end{aligned}$$

Then, since $0 \leq a_{ij} \leq 1$,

$$K - m \leq \Delta^B_i \leq K + m. \tag{8}$$

We now wish to "fill the deficits" Δ^B_i by changing only the values (in ξ^B) of variables with small reduced costs. Then we will use Lemma 7 to show that the solution ξ so constructed is close to optimal. We change the ξ^B by making cyclic changes within chosen blocks of m columns. To this end, we make the following definitions. Let s be any integer. We define an *s-good* block to be a set of m columns j, as follows. For ease of notation, let us assume the block comprises columns $1, 2, \ldots, m$, and that subscript arithmetic is modulo m. Let us write $\tilde{c}_{ij} = c_{ij} - u_i a_{ij}$. Then the block must be such that, for $1, 2, \ldots, m$

(a) $\xi_{ii}^B = 1$. (The changes we intend to make are $\xi_{ii} = 0$, $\xi_{i-1,i} = 1$.)

(b) $a_{ii} \leq \eta/20$, $a_{i,i+1} \in [a_{ii}, a_{ii} + \eta/20]$.

(c) $c_{ji} \leq \frac{1}{2}$, $j \neq i, i-1$, $\frac{3}{5} < c_{ii} \leq \frac{4}{5}$, and

$$c_{i-1,i} \in [\tilde{c}_{ii} + u_{i-1}a_{i-1,i} - s\epsilon, \tilde{c}_{ii} + u_{i-1}a_{i-1,i} - (s-1)\epsilon]$$

These imply, by straightforward calculations, the following conditions for column i in the block,

(i) $\tilde{c}_{ki} \leq c_{ki} \leq \frac{1}{2}$ for $k \neq i, i-1$. (The LP solution must not pick x_{ki}.)

(ii) $\tilde{c}_{ii} > \frac{3}{5} - \frac{1}{10} = \frac{1}{2}$. (The LP solution must pick x_{ii}.)

(iii) $(s-1)\epsilon \leq \tilde{c}_{i-1,i} - \tilde{c}_{i,i} \leq s\epsilon$. (The LP solution picks x_{ii}, but $x_{i-1,i}$ is "nearly as good". Observe also that conditions (i)–(iii) imply that $\tilde{c}_{ij} = \tilde{c}_{ij} - \tilde{c}_{ii}$ for all i, j.)

(iv) $a_{i,i+1} - a_{ii}$ is uniform in [0,η/20] independently, even conditional on a_{ii}. (We need this since we wish to argue relative to the optimal basis for $LP(b')$, and a_{ii} is then conditioned by feasibility of the solution.)

(v) $[\tilde{c}_{ii} + u_{i-1}a_{i-1,i} - s\epsilon, \tilde{c}_{ii} + u_{i-1}a_{i-1,i} - (s-1)\epsilon] \subseteq [0,1]$ for $s \leq 1/2\epsilon = \Omega(n/\log n)$. (We require that the probability that $c_{i-1,i}$ falls in this range depends only on its width, and not its location.)

Now the probability that a given column j can be the ith column in an s-good block is clearly *a priori* at least

$$\left(\frac{1}{2}\right)^{m-2} \times \frac{1}{5} \times \left(\frac{\eta}{20}\right)^2 \times \epsilon \;>\; \frac{\epsilon}{M},$$

independently of i or j.

Suppose now that, for *any* dual feasible basis D, we try to construct s-good blocks. The determination of the u_i conditions only m columns, by Lemma 4. We now scan the remaining $(n-m)$ columns from left to right, attempting to form good blocks one column at a time. There is one problem. The circularity of our conditions for the columns of the blocks implies that we cannot test column 1 in the block until column m is known. For this reason we have to proceed slightly more carefully, as follows. We denote the elements of the columns as in the above conditions, but the reader should observe that we are in fact examining a general column j for potential addition to the current incomplete block. We scan to select the first column, checking conditions (a)–(c). However, instead of checking condition (b) on $a_{i-1,i}$ (i.e. a_{m1}) (since this is currently impossible) we merely check the necessary condition that $a_{m1} \leq \eta/10$. Note that we also check condition (c) involving a_{m1}, but the choice of our ranges implies that this does not further condition a_{m1}. It merely locates the point (c_{m1}, a_{m1}) in a small parallelogram whose projection is uniform on the a_{m1}-axis. Having chosen column 1 we scan to add columns 2,...,$m-1$, in each case checking conditions (a)–(c). There is no difficulty since the required quantities are all known at the time of checking. We now select column m. First

we check all conditions except that in (b) involving a_{m1}. We will check condition (c) involving a_{mm}, but this does not condition it further than $a_{mm} \leq \eta/20$, by the same argument used for a_{m1}. Let us call a "block" which has passed all tests up to this point a *candidate* block. Now the point $\pi = (a_{mm}, a_{m1})$ is uniformly distributed in the rectangle $\Pi = [0, \eta/20] \times [0, \eta/10]$. We now test condition (b) on a_{m1}. This requires that the point π lies in a parallelogram whose area is half that of Π. Thus the probability that a candidate block is s-good is $\frac{1}{2}$. If the candidate block is not good, we reject it completely and start afresh. This may appear drastic, but to do otherwise introduces extra conditioning, and the above line of argument fails.

Note that, as we form candidate blocks, the probability of a given column being successfully added is still (independently) at least ϵ/M, since we check some subset of (a)–(c). However some candidate blocks will not be good. Now the expected number of successful additions is $(n-m)\epsilon/M > 4mk$ for large n. Thus

$$\begin{aligned} \Pr(\text{Under } 3k \text{ candidates formed}) &= \Pr(\text{Under } 3km \text{ successful additions}) \\ &< e^{-\frac{1}{8}mk} \\ &< n^{-2m^3}, \end{aligned}$$

for large n, using Hoeffding's inequality [4, 7]. Then, since each candidate is good with probability $\frac{1}{2}$.

$$\begin{aligned} \Pr(\text{Under } k \text{ } s\text{-good blocks formed}) &< e^{-k/18} + n^{-2m^3} \\ &< 2n^{-m^2} \end{aligned}$$

for large n, again using Hoeffding's inequality.

Now, since there are less than n possible values of s, and only $O(n^m)$ dual feasible bases D to consider, we see that for large n,

$$\Pr(\exists\ D, s : \text{less than } k \text{ } s\text{-good blocks exist}) < n^{-\frac{1}{2}m^2}.$$

For a given s-good block, let $\delta_i = a_{i,i-1} - a_{ii}$ for $i = 1, 2, \ldots, m$. These give the changes in the left hand sides of the constraints in (1) when the cyclic changes $\xi_{ii} = 0$, $\xi_{i-1,i} = 1$ are made to the variables in the block. If we have k s- good blocks we denote the corresponding values by δ_{ir} for $r = 1, 2, \ldots, k$. By construction the δ_{ir} are distributed uniformly and independently in $[0, \eta/20]$. Let

$$Z_{ir} = \left(\frac{\delta_{ir} - \eta/40}{\eta/40}\right)\sqrt{3}.$$

Z_{ir} is uniform in $[-\sqrt{3}, +\sqrt{3}]$ with mean 0, variance 1. Thus

$$\begin{aligned} \Delta_i^B - \sum_{r=1}^{k} \delta_{ir} &= \Delta_i^B - \left(\frac{\eta}{40\sqrt{3}}\right)\sum_{r=1}^{k} Z_{ir} - \frac{k\eta}{40} \\ &= \Gamma_i - \frac{\eta}{40\sqrt{3}}\sum_{r=1}^{k} Z_{ir}, \end{aligned}$$

where $\Gamma_i = \Delta_i^B - (k\eta/40) = K - (k\eta/40) + O(1) = O(1)$, using (8) and the definitions of k, K. Let $\Phi_i = 40\sqrt{3}\Gamma_i/\eta$. Now we use the following Lemma, which is a (modified) restatement of Lemma 3.4 of [2].

Lemma 8 *Let Z_{ir}, $i = 1,\ldots,m$, $r = 1,\ldots,k$ be bounded i.i.d. random variables with bounded density, expectation 0 and variance 1. Let $I_i = [\Phi_i - \theta, \Phi_i]$, where $\theta = \frac{(\pi k/2)^{(m+1)/2m}}{2^{k/m+1/2}}$ and $\Phi_i = O(k^{1/4})$, be m intervals. Let $q_m = \frac{1}{2(1+(2/\sqrt{3})^m)}$, and let S be an arbitrary subset of $\{1,2,\ldots,k\}$ with $|S| = \lfloor k/2 \rfloor$. Then, for large k,*

$$\Pr(\exists S : \sum_{r\in S} Z_{ir} \in I_i, i = 1,\ldots,m) \geq q_m.$$

□

Applying this to our situation, we see that with probability at least q_m we can choose $\lfloor k/2 \rfloor$ s-good blocks to change which will leave the deficit in each left hand side in (1) at most $\eta\theta/40\sqrt{3}$. Substituting for k, this is less than (say) n^{-10m}, for large n. Now apply Lemma 7. By using only s-good blocks, with probability at least q_m we can construct an integer solution ξ to (1) with objective value z_{IP} satisfying

$$0 \leq z_{LP} - z_{IP} \leq ks\epsilon + 2mn^{-10m}/\eta < s\alpha(\log n)^2/n, \tag{9}$$

for large n, with $\alpha = 2^{m+20}m^5/\eta^2$, say. Now we try the s-good sets for $s = 1,2,\ldots,t$, seeking to find s so that (9) is satisfied. (Note that t can be as large as $n/(\log n)^2$.) Let $p = 1 - q_m$. Each "trial" is independent (since it involves only the δ_{ir}). The probability that we fail t times is thus at most p^t, as required. This proves the first statement of Theorem 1. The second is derived from it as follows.

Let $t = \lceil 2\log n/\log(1/p) \rceil$, say. Using a simple bound on expectation, and the fact that $z_{LP} - z_{IP} \leq n$ always, we have

$$\begin{aligned} 0 \leq E(z_{LP} - z_{IP}) &\leq (\alpha(\log n)^2/n)\textstyle\sum_{s=1}^{t} sp^{s-1} + np^t \\ &< (\alpha/(1-p)^2)(\log n)^2/n + 1/n \\ &< (2\alpha/(1-p)^2)(\log n)^2/n, \end{aligned} \tag{10}$$

for large enough n. The final statement of Theorem 1 follows from this and Lemma 9 below. Let

$$C_1 = \{(i,j) : \bar{c}_{ij} \in (0, 2\log n/n]\}.$$

It is shown in Lemma 9 that

$$E(|C_1|) < 4m^2 \log n. \tag{11}$$

Now, there are at most $2m$ fractional basic variables by Lemma 3. Thus clearly

$$\mathcal{N} \leq 2m + |C_1| + \frac{(z_{LP} - z_{IP})}{2\log n/n}.$$

Taking expectations of both sides in this inequality, the theorem now follows using (10) and (11). □

4 Proof of Theorem 2

Let $\eta > 0$ be given, and let $0 \leq \tau \leq 1$ be the required failure probability. Assume n is large enough that

$$\Pr(\,(1) \text{ is } \eta\text{–feasible}) \geq 1 - \tau/2.$$

Now consider the following algorithm.

Algorithm OPT_τ

(1) $t \leftarrow \lceil \log(\tau/2)/\log p \rceil$.

(2) Solve $LP(b)$ giving x^B. Modify this to ξ^B.

(3) Let $T = \{(i,j) : \xi^B_{ij} = 0, 0 \leq \bar{c}_{ij} \leq t\alpha(\log n)^2/n\}$, where $\bar{c}_{ij}$ are the reduced costs for x^B.

(4) Compute $\min_{S \subseteq T} c\xi_S$, for all S and ξ_S such that

(a) there is at most one $(i,j) \in S$ for all j.

(b) $\sum_{(i,j)\in S} \bar{c}_{ij} \leq t\alpha(\log n)^2/n$.

(c) ξ_S is feasible in (1), where ξ_S is obtained from ξ^B by $\xi_{ij} \leftarrow 1$, $\xi_{kj} \leftarrow 0$ $(k \neq i)$, for each $(i,j) \in S$.

The minimising S at termination (if defined), gives an optimal solution $\xi_0 = \xi_S$ to (1). This follows directly, since the search accounts for all feasible integer solutions having objective value at most $z_{LP} + t\alpha(\log n)^2/n$. Failure because (1) is not η-feasible has probability at most $\tau/2$. The probability that S is undefined is, by Theorem 1, less than p^t, which is at most $\tau/2$ by choice of t. Thus it remains to analyse the running time of OPT_τ.

Fix any dual feasible basis D, and let

$$\begin{aligned} J_l &= (\,2(l-1)\log n/n, 2l\log n/n\,], \\ C_l &= \{(i,j) : \bar{c}_{ij} \in J_l\}. \end{aligned}$$

(C_1 has already been used in §3.)

Lemma 9 $\Pr(|C_l| \geq 4m^2 \log n) \leq n^{-2m^2/3}$.

Proof For any column j not determining the u_i, we have

$$\begin{aligned} E(\text{number of } (i,j) \in C_l) &\leq E(\text{number of pairs } i,r : \tilde{c}_{ij} - \tilde{c}_{rj} \in J_l) \\ &\leq m(m-1)\Pr(c_{1j} \in [\lambda, \lambda + 2\log n/n]), \\ &\quad \text{for } \lambda = u_1 a_{1j} + c_{2j} - u_2 a_{2j}, \\ &\leq 2m^2 \log n/n. \end{aligned}$$

Hence, using Hoeffding's inequality,

$$\Pr(|C_l| > 4m^2 \log n) \leq e^{-2m^2 \log n/3} = n^{-2m^2/3}.$$

□

Since there $O(n^m)$ dual feasible bases, and we are interested in only $O(\log n)$ values of l, it follows from Lemma 9 that we may assume $|C_l| \leq 4m^2 \log n$ for all l in the optimal basis. We examine the search in OPT_r relative to the sets C_l. To these we have only to add the (less than $2m$ by Lemma 3) fractional variables which have been rounded down. The argument is now similar to that in [2].

Thus let $R_r = \bigcup_{l=2^{r-1}}^{2^r-1} C_l$ $(r = 1, 2, \ldots)$. We clearly need only consider S for which

$$|S \cap R_r| \leq \left\lfloor \frac{t\alpha(\log n)^2/n}{(2^r - 2)\log n/n} \right\rfloor = \left\lfloor \frac{t\alpha \log n}{(2^r - 2)} \right\rfloor = \rho_r,$$

say, if $r > 1$. Note that $\rho_r = 0$ for all large enough r. Thus, with probability $1 - o(1)$, we need to check at most N sets S, where

$$N = 2^{2m} \times 2^{4m^2 \log n} \times \prod_{r=2}^{\infty} N_r,$$

sets S, where

$$N_r = \sum_{r=0}^{\rho_r} \binom{\lfloor 4m^2 \log n 2^{r-1} \rfloor}{r}.$$

Now define $r_0 = \max\{r : \frac{t\alpha}{2^r - 2} \geq m^2 2^{r-1}\}$, and note that r_0 is independent of n. For all r, clearly $N_r \leq 2^{4m^2 \log n 2^{r-1}}$. However, for $r \geq r_0$, using well-known bounds on binomial coefficients, we have the better bound

$$N_r \leq 2\left(\frac{e\, 4m^2 \log n 2^{r-1}}{\rho_r}\right)^{\rho_r} \leq 2^{\gamma(r/2^r)\log n},$$

after a straightforward calculation, for large n and suitably chosen γ (independent of n and r). Hence

$$N \leq 2^{2m + 4m^2 \log n + \sum_{r=2}^{r_0} 2^{r+1} m^2 \log n + \sum_{r=r_0+1}^{\infty} \gamma(r/2^r)\log n} = n^{d-1},$$

for a suitable d. Since each set S takes $O(n)$ time to process, the theorem follows. □

5 Remarks

(1) For convenience in the development, we have stated all our results for large n, but obviously they can be modified to hold for all n (simply by enlarging the constants if necessary).

(2) We claim only that our results hold for constant m, but it would appear that generally they can be modified to remain true if m grows slowly enough with n. However, the running time is (at least) $\Omega(C^{m^2 \log n})$, for some constant C. Thus the algorithm is not provably polynomial time for non-constant m.

(3) The approach we have developed here appears to be applicable to some related problems, for example the multiple-choice multidimensional knapsack problem.

References

[1] J. F. Benders and J. A. Vannunen (1983), A property of assignment type mixed integer programs, *Operations Research Letters* **2**, 47–52.

[2] M. E. Dyer and A. M. Frieze (1989), Probabilistic analysis of the multidimensional knapsack problem, *Mathematics of Operations Research* **14**, 162–176.

[3] A. Goldberg and A. Marchetti-Spaccamela (1984), On finding the exact solution of a zero-one knapsack problem, *Proceedings of the 16th Annual ACM Symposium on the Theory of Computing*, 359–368.

[4] W. Hoeffding (1963), Probability inequalities for sums of bounded random variables, *Journal of the American Statistical Association* **58**, 13–30.

[5] K. O. Jörnsten and M. Nasberg (1986), A new Lagrangian relaxation approach to the generalised assignment problem, *European Journal of Operations Research* **27**, 313–323.

[6] G. S. Lueker (1982), On the average difference between the solutions to linear and integer knapsack problems, in *Applied Probability–Computer Science: The Interface* Vol. I, (Eds. R. L. Disney and T. J. Ott), *Progress in Computer Science*, Birkhäuser, Boston, pp. 489–504.

[7] C. J. H. McDiarmid (1989), On the method of bounded differences, in *Surveys in Combinatorics, 1989* (Ed. J. Siemons), London Mathematical Society Lecture Notes 141, pp. 148–188.

[8] G. T. Ross and R. M. Soland (1975), A branch-and-bound algorithm for the generalised assignment problem, *Mathematical Programming* **8**, 91–103.

On the clique-rank and the coloration of perfect graphs

Jean FONLUPT and Andràs SEBÖ
IMAG-ARTEMIS ,CNRS, Université de Grenoble.
BP 53x 38041 Grenoble France

Abstract: We are investigating the *clique-rank (linear rank of the ω-cliques)* of graphs and its relation to the optimal colorations of perfect graphs.

First, we observe a simple characterization of perfectness with an inequality about the clique rank. Then we notice that the extremal graphs with respect to this inequality are exactly the uniquely colorable perfect graphs (perfect graphs that have exactly one optimal coloration), and that a *good characterization for the unique colorability* can also be derived with the help of the clique-rank.

After this, we would like to understand the significance and the use of uniquely colorable perfect graphs (perfect graphs that have exactly one optimal coloration) for the perfect graph conjecture. In lack of a general *combinatorial* good characterization of unique colorability we state a conjecture for such a characterization, study its connection with other conjectures and theorems (including the Perfect Graph Conjecture). The validity of our conjecture, even for special cases, would imply new classes of perfect graphs. After settling it for $\omega=3$, we deduce the Perfect Graph Conjecture for diamond-free graphs, a result of Partasarathy, Ravindra and Tucker. This study relies again on the clique rank.

Finally, we show how a simply stated combinatorial algorithm for coloring perfect graphs can be designed with the help of the clique rank, an algorithm that unfortunately depends exponentially on ω. However, this algorithm is efficient for some classes perfect graphs, including "K_4-e"-free perfect graphs treated earlier by Tucker. Its performance in general is $O(n^{\omega+1})$, which, if ω is small, is better than the ellipsoid method, and than other known algorithms .

1. Introduction

Let G = (V(G), E(G)) be an undirected graph. A clique of G is a subset of pairwise adjacent vertices. Let $\omega(G)$ be the maximum cardinality of a clique. A *k-coloration* of G is a partition of V(G) *into k stable sets.* The chromatic number $\chi(G)$ is the minimum number of stable sets partitioning V(G). A graph is called *perfect* if $\omega(G')=\chi(G')$ for every induced subgraph G' of G. G will be called *uniquely colorable*, if it has exactly one $\omega(G)$-coloration.

A chordless cycle is a cycle which is an induced subgraph. Berge's Perfect Graph Conjecture (PGC) states that *a graph is perfect if and only if it does not contain an odd chordless cycle or the complement of an odd chordless cycle as an induced subgraph .*

Let A(G) be the incidence matrix of the cliques of G and B(G) be the incidence matrix of the $\omega(G)$-cliques of G. (B(G) is a submatrix of A(G)).The linear rank of the incidence (characteristic) vectors of the $\omega(G)$-cliques of G will be called the *clique rank* of G, and will be denoted by r(G).Note that r(G) = r(B(G)).Sets and their incidence vectors will often not be distinguished. A maximum set of linearly independent $\omega(G)$-cliques (that is a set that generates linearly all $\omega(G)$-cliques) will be called a *clique-base*; if $\omega(G)$ =3, then it will be called a *triangle-base*; their span, that is the span of all $\omega(G)$-cliques will be called the *clique-space* (for $\omega(G)$ =3 *triangle-space*).

Let us recall a basic result about perfect graphs :

(1.1) The following statements are equivalent :
(i) A graph is perfect.
(ii) The stable set polytope $P(G)=\{x : x \geq 0; A(G)\, x \leq 1\}$ is integral, that is the set of its extreme points is the set of incidence vectors of the independent sets of G.
(iii) (i) or (ii) hold for the complement of G.

The equivalence of (i) and (ii) is an immediate consequence of Lovàsz's main "blowing Lemma" in the proof of the Perfect Graph Theorem, a remark implicit in Lovàsz (1972), explicit in Chvàtal (1975) and conjectured by Fulkerson (1970). The equivalence of (iii) with the rest is just the Perfect Graph Theorem (Lovàsz 1972). (1.1) will be extensively used in the sequel.

Many results related to perfect graphs are based on this polyhedral characterization. Padberg's results (1974) on critically imperfect graphs are among them. We just mention two of the properties he proved, for later convenience:

(1.2) If G is a criticlally imperfect graph , then
a. r(G)=n
b. $G(V(G)\setminus\{v\})$ is uniquely colorable for every $v \in V(G)$.

(An imperfect graph is critically imperfect if any proper induced subgraph is perfect.) Another interesting result, that follows from Fulkerson (1971) is that there exists a polynomial algorithm for finding a minimum coloring of G provided that $\omega(G)$ is fixed. Again, this algorithm uses some powerful tools of linear algebra. Note however that the method proposed by Fulkerson is not combinatorial.

Similar remarks could be said about Grötschel Lovàsz and Schrijver's algorithm for coloring perfect graphs, the only known polynomial algorithm for this problem.

On the other hand many questions related to perfect graphs can be solved using purely combinatorial arguments; Berge's conjecture itself has a purely combinatorial flavor.

Our goal in this paper is to connect these two possible approaches; more specifically, we use linear algebra in order to get combinatorial methods and structures for the coloring problem of perfect graphs. In particular, we are interested in uniquely colorable perfect graphs which play a crucial role in perfect graph theory by the above mentioned result of Padberg. We shall see that from the point of view of linear algebra they are the "most saturated" perfect graphs.

In Section 2 we prove some general results based on the *clique rank* and the *clique space* of a graph. In particular, we show how perfectness can be characterized with the help of the clique rank. From the observation that uniquely colorable perfect graphs are those with maximum clique rank, we deduce a *good characterization of unique colorability* for perfect graphs.

In Section 3 we state a conjecture about a stronger, combinatorial characterization of unique colorability, and its relation to other conjectures and results. We would also like to explain why the special cases with fixed ω are interesting.

In Section 4 we prove the main conjecture for $\omega=3$, applying some algebraic considerations. This is a common generalization of the perfect graph conjecture for diamond-free and 3-chromatic perfect graphs, known earlier from the work of Parthasarathy and Ravindra (1979), and Tucker (1984), (1987a,b).

In the last Section 5 we show how the relation of the clique rank and colorations can be used for coloring algorithms.

2. The clique rank and the clique space

In this section we wish to make clear some simple connections between the clique rank and the colorations of perfect graphs.

Let us first define an upper bound for the clique rank of a perfect graph. We shall see that this bound is tight (equality is satisfied) exactly for uniquely colorable graphs (Theorem 2.3 below).

Proposition 2.1 If G is perfect, then $r(H) \leq n-\omega(H)+1$ for every induced subgraph H of G.

Proof: Let H be an induced subgraph of G. Since H is perfect there are $\omega(H)$ stable sets which partition V(H). These $\omega(H)$ vectors are clearly linearly independent, and are solutions of the linear system $B(H)x=1$. Thus $r(H) \leq n-\omega(H)+1$.

The following corollary will be important from an algorithmic point of view:

Corollary 2.2 : Let G be a perfect graph whose edge-set is non-empty.
a.) If $\mathfrak{K}$ is a set of linearly independent $\omega(G)$ -cliques, then there exists $v \in V(G)$ such that v is contained in at most $\omega(G)-1$ elements of $\mathfrak{K}$.
b.) There exists $v \in V(G)$ such that $r(G) \leq r(G-v)+\omega(G)-1$.

Proof: a)Let $\mathfrak{K}(v) := \{K \in \mathfrak{K} : v \in K\}$. If each vertex were contained in at least $\omega(G)$ elements of $\mathfrak{K}$, then we would have $\omega(G)|\mathfrak{K}| = \Sigma\{|\mathfrak{K}(v)| : v \in V(G)\} \geq \omega(G)\ |V(G)|$, whence $|\mathfrak{K}| \geq |V(G)|$. On the other hand $|\mathfrak{K}| \leq |V(H)|-\omega(H)+1$, and since $\omega(H) \geq 2$, this is a contradiction.
b.) Take the vertex v given by a). $r(G-v) = r(G) - |\mathfrak{K}(v)| \geq r(G) - (\omega(G) - 1)$, as we claimed

The following theorem summarizes the main remarks we made so far. It will be used to prove non-perfectness if the coloring algorithms fail.

Theorem 2.3: The following statements about the graph G are equivalent:
(i) G is perfect.
(ii) For every induced subgraph H of G: $r(H) \leq n-1$, provided H contains an edge.
(iii) For every induced subgraph H of G: $r(H) \leq n-\omega(H)+1$.

Proof: "(i)⇒(iii)" is just Proposition 2.1 .
If H contains an edge, then $\omega(H) \geq 2$, whence "(iii)⇒(ii)" is obvious.
"(ii)⇒(i)" is just (1.2) a., and the Theorem is proved.

Note that the bounds in (ii) and (iii) are equal if and only if $\omega=2$, and the gap for $\omega>2$ is quite mysterious. (Has this particular role of $\omega=2$ something to do with the same role of this case in the perfect graph conjecture ?)

Theorem 2.4: Let G be a perfect graph. G is uniquely colorable if and only if

$$r(G) = n-\omega(G)+1.$$

Proof: Let $P(G)= \{x\geq 0 : A(G)x\leq 1\}$ be the stable set polytope of G}.
For basic knowledge about polyhedra we refere to Schrijver (1986). Recall however that a d -dimensional face of a full dimensional polyhedron is the intersec-tion of n-d facets having a set of linearly independent coefficient vectors.

P(G) is obviously full dimensional. L et F be the face $F=\{x\in P(G) : B(G)x=1\}$ of P(G). The extreme points of F are exactly the stable sets which intersect every $\omega(G)$-clique. In other words *a stable set S belongs to F if and only if it occurs as a color class in some $\omega(G)$-coloration.* If S has this property it will be called a *color-class.* Since F is the intersection of the facets defined by the rows of B(G), dim(F)+ r(G)=n. (Recall that r(G)=r(B(G)).)

G is uniquely colorable if and only if the number of color classes is $\omega(G)$, which is true if and only if $\dim(F)=\omega(G)-1$. (A color class different from the classes of a given coloration, is also linearly independent of them.) . Thus the theorem is proved.

This is obviously a good characterization of unique colorability: an $\omega(G)$- coloration, and a set of $n-\omega(G)+1$ linearly independent $\omega(G)$-cliques of a graph on n vertices proves that the graph is uniquely $\omega(G)$-colorable. This good characterization can be presented in the following combinatorial way:

Suppose $\omega(G)\geq 3$. (The case $\omega(G)\leq 2$ is trivial.) $a,b\in V(G)$ will be said to be equivalent, in notation $a\sim b$ if there exist multisets (a multiset is a set whose elements have non-negative integer multiplicities showing the number of times they are contained in the multiset) $\mathfrak{K}_1$ and $\mathfrak{K}_2$ of $\omega(G)$-cliques such that $\mathfrak{K}_1(x) = \mathfrak{K}_2(x)$ holds if and only if $x \notin \{a,b\}$, where $\mathfrak{K}_i(x) := |\{K\in\mathfrak{K}_i: x\in K\}|$.

The following theorem is a combinatorial version of theorem 2.4, and we also included a characterization for a and b to have the same color in every coloration, even if G is not uniquely colorable:

Theorem 2.5: Let G be a perfect graph with $\omega(G)\geq 3$.

- The above defined relation is an equivalence relation;
- $a\neq b\in V(G)$ have the same color in all the ω-colorations of G if and only if they are equivalent.;
- G is uniquely colorable if and only if this equivalence relation has $\omega(G)$ classes.

Proof: One can remain within the framework of "combinatorial " arguments for the trivial parts. By simple counting arguments one can prove the following about $a, b\in V(G)$, $a\sim b$ and about the families of cliques in the definition of the equivalence:

1. $|\mathfrak{K}_1|=|\mathfrak{K}_2|$.
2. $\mathfrak{K}_1(a) - \mathfrak{K}_2(a) = -(\mathfrak{K}_1(b) - (\mathfrak{K}_2(b))$ (see the notations of Corollary 2.2)
3. " $\sim$" is an equivalence relation

Let us now check the trivial if parts of the statements in a combinatorial way. Let $a\sim b$, and let $\mathfrak{K}_i$ (i=1,2) be the $\omega(G)$-cliques which prove this equivalence. Let $S_1,\ldots, S_\omega$ be a coloration, and suppose indirectly $a\in S_1$, $b\notin S_1$. $|\mathfrak{K}_i|=\sum_{x\in S_1}\mathfrak{K}_i(x)$,

Since $\mathfrak{K}_1(x) = \mathfrak{K}_2(x)$ if $x\neq a$, but $\mathfrak{K}_1(a)\neq\mathfrak{K}_2(a)$, we can conclude $|\mathfrak{K}_1|\neq|\mathfrak{K}_2|$. Let now S_2 be a color class of the above coloration which contains neither a nor b. (There exists such a color class since $\omega(G)\geq 3$.) We have now, like above, $|\mathfrak{K}_i|=\sum_{x\in S_2}\mathfrak{K}_i(x)$, which implies now $|\mathfrak{K}_1|=|\mathfrak{K}_2|$, a contradiction. Thus a and b have the same color in every coloration, and the if part of both statements follows immediately.

To prove the essential only if part we shall use some linear algebra in a similar way as in the proof of Theorem 2.4. Suppose $a, b\in V(G)$ have the same color in every $\omega(G)$-coloration of G, and define the vector w with w(a)=1, w(b)= -1 and w(v)=0 if $a\neq v\neq b\in V(G)$. By the assumption on a and b, ws=0 holds for the characteristic vector s of every color class. (A color class contains neither a nor b or both a and b.) Since G is perfect the color classes generates the set of all the solutions of the system B(G)x=1. Thus wx=0 is satisfied for every solution of the system B(G)x=1, and consequently w belongs to the linear space generated by the rows of B(G): $w=\sum\lambda_i K_i$. Let d be their smallest common denominator of the

λ_i. The families $\{K_i : \lambda_i>0\}$ and $\{K_i : \lambda_i<0\}$ where the multiplicity of clique K_i is $|d\lambda_i|$ prove a~b, and the proof is finished.

Note however, that despite the tanslation into a combinatorial language, the above solution is actually algebraic. Unfortunately it does not say enough about the structure of uniquely colorable perfect graphs (see Section 3).

For $\omega=3$ we shall present in Section 4 a "more purely" combinatorial characterization.

3. Some conjectures

One of the most promising trials to approach the perfect graph conjecture has been Tucker's analysis of uniquely colorable perfect graphs, and their relation to the the perfect graph conjecture. Let us summarize Tucker's approach:

A.) Look for a good characterization of the unique colorability of perfect graphs: in a uniquely colorable perfect graph a *combinatorial* "forcing procedure" of the unique

$\omega(G)$-coloration should be constructed. (Theorem 2.5 gives already some forcing procedure based on linear algebra.)

B.) Suppose that a forcing procedure is applied to the (not necessarily perfect) graph G, and that a certain vertex is forced to have an $\omega(G)+1$-th color. Proving that in this case either an $\omega(G)+1$-clique or an odd chordless cycle or the complement of an odd chordless cyclecan be found, is equivalent to the PGC, as it is explained below. (It is obviously implied by the PGC.)

Let us explain how an appropriate answer to A.) and B.) would imply the Perfect Graph Conjecture. Let G be critically imperfect. By (1.2) G-v is uniquely $\omega(G)$colorable: apply now A.) for coloring G-v. Clearly, v must be colored with an $\omega(G)+1$-th color. Now we can apply B.) to find a hole or an antihole, and the Perfect Graph Conjecture is "proved".

Some results have been reached earlier in the direction of B.). Chvàtal's result (1984) can be interpreted as showing the existence of a hole or antihole under a strengthening of the unique colorability condition. See also Tucker (1983) for some generalizations. On the other hand there has not been anything done about A.), besides some conjectures of Tucker.

Let us state a conjecture for a combinatorial forcing of unique colorability:

Let v and w be two non-adjacent vertices of a graph G. We say that v and w are *forced* vertices if there exist two $\omega(G)$-cliques K_1 and K_2 such that $\{v\}= K_1 \setminus K_2$ and $\{w\}= K_2 \setminus K_1$.

Obviously, if G is perfect, in any optimal coloration the same color is assigned to v and w.

Starting from a graph G let H be the graph obtained by repeated identification of forced vertices. (Note that this operation does not necessarily preserve perfectness.)

Conjecture 1: A perfect graph is uniquely colorable if and only if H is a clique of size $\omega(G)$.

The sufficiency of this condition is obvious. In the following section we prove it for K_4-free graphs, a case whose significance is explained by Theorem 3.2 below. The validity of this Conjecture would be especially interesting for classes of graphs for which the Perfect Graph Conjecture is not proved yet.

Recently, Markossian and Gasparian (1988) proved a theorem in the direction of B.) using a similarly looking forcing procedure to the above defined. However, their forcing procedure is somewhat more particular. A proof of the Perfect Graph Conjecture using the above conjecture would in particular include their result and the main result of Tucker (1983). Such a proof may turn out to be less difficult than the conjecture itself.

Tucker (1983) had already a (more complicated) conjecture on unique colorability. We state without proof the equivalence of the two conjectures:

Theorem 3.1: The above conjecture is valid if and only if Tucker's conjecture is valid.

In fact we cannot even prove the existence of one forcing step:

Conjecture 2: If G is a uniquely colorable perfect graph, then it contains two $\omega(G)$-cliques whose intersection is an $\omega(G)$-1-clique.

Finally let us mention a more modest goal than the PGC which also makes use of unique colorability:

Theorem 3.2 *If $\mathcal{G}$ is a class of graphs closed under taking induced subgraphs, and for some integer k every perfect graph in $\mathcal{G}$ with $\omega=k$ has at least two colorations and every graph in $\mathcal{G}$ with $\omega<k$ is perfect, then every graph in $\mathcal{G}$ is perfect.*

This statement is a simple consequence of the above mentioned result of Padberg:

Proof of Theorem 3.2: Suppose for $\mathcal{G}$ and k the conditions of the theorem are satisfied, but the statement is not true: let $G\in\mathcal{G}$ be not perfect and $|V(G)|$ minimum among such graphs. Since $\mathcal{G}$ is closed under taking induced subgraphs, G contains a critically imperfect induced subgraph which is in $\mathcal{G}$, and thus G itself is such a graph.

By our assumption on $\mathcal{G}$, $\omega(G)\geq k$. Take an arbitrary $v\in V(G)$. Since G is critically imperfect, $\omega(G-v)=\omega(G)\geq k$. Clearly, G-v is perfect, and by Padberg's result it is UC, whence the union of k color classes of its unique ω-coloration induces a graph H which is also UC. H is perfect, H belongs to $\mathcal{G}$, $\omega(H)=k$, and thus, by the assumption, it has two colorations, a contradiction.

Note that *k plays a role only in the condition of Theorem 3.2 , and it is not necessarily present in the definition of the class* $\mathcal{G}$. Thus, information about uniquely k-colorable perfect graphs can lead to new classes of perfect graphs. This gives some motivation to look at the case k=3 in more details.

4. The case $\omega=3$

The idea of Theorem 3.2 has already been exploited for k=2: clearly, *a bipartite graph is UC if and only if it is connected.* The proof of Giles, Trotter and Tucker (1984) of the PGC for $K_{1,3}$-free graphs repeatedly uses this fact.

For k=3 the following proof of the PGC for diamond-free graphs gives an example to the use of this idea. A *diamond* is a graph isomorphic to $K_4\backslash e$. *Diamond-free* graphs are graphs which do not contain a diamond as an induced subgraph. The following result can be checked through the decomposition procedure for diamond-free perfect graphs of Fonlupt and Zemirline (1988).

Theorem 4.1: A diamond-free non-bipartite perfect graph has at least two colorations.

This is Conjecture 2 for $\omega=3$. *Theorem 3.2 and Theorem 4.1 immediately imply the Perfect Graph Conjecture for diamond-free graphs,* a result of Parthasarathy and Ravindra (1979). The results of this section led us to a proof of Conjecture 1 as well, for $\omega=3$.

Despite the combinatorial nature of the results in this section, their proofs are based on an algebraic observation. In order to state this let us introduce some notations.

r_2 will denote the rank function over GF(2), and $r_2(G)$ will denote the rank over GF(2) of the set of characteristic vectors of all $\omega(G)$-cliques of a graph. Note that the incidence vectors of a family of subsets of V(G) are linearly independent over GF(2) if and only if for all subfamilies (including the original family) there exists $v\in V(G)$ which is contained in an odd number of sets belonging to the subfamily.

The essential part of Theorem 4.2 below is that there exists a basis for the triangle space over GF(2), which is also a basis over the rational numbers:

Theorem 4.2 : Let G be a 3-colorable perfect graph.

a.) $r_2(G)=r(G)$.

b.) If the integer vector z is in the triangle space and t is the 0-1vector such that $t\equiv z \pmod 2$, then t is in the triangle space over GF(2) as well.

Proof: Note first that $r_2(\mathcal{T}) \le r(\mathcal{T})$ is obvious. To prove the equality, let $k:=n-r_2(\mathcal{T})$. It is well known from linear algebra that there are k vectors $t^1,\dots,t^k \in\{0,1\}^n$ linearly independent over GF(2), each of which is orthogonal to $\mathcal{T}$, that is: $t^i(T)\equiv 0 \pmod 2$ for i=1,...k and for every $T\in\mathcal{T}$. We shall construct linearly independent vectors $z^1,\dots,z^k \in \mathbb{Z}^n$ such that $z^i(T)=0$ (for i=1,...k and for all $T\in\mathcal{T}$). This will prove the statement, because then $r(\mathcal{T})\le n-k=r_2(\mathcal{T})$.

If $T\subseteq V(G)$ is a triangle of G, then it is a linear combination of elements of $\mathcal{T}$, whence $t^i(T)\equiv 0 \pmod 2$ (i=1,...k). This means that the set $X_i:=\{v\in V(G): t^i(v)=1\}$

intersects every clique in an even number of elements. In particular, it does not contain any triangle: in other words $\omega(G_i)\leq 2$ for the graph G_i induced by X_i. Since G is perfect, G_i is bipartite. Denote the classes of a bicoloration of G_i by A_i and B_i. Let $z^i(v):=1$ if $v\in A_i$ and -1 if $v\in B_i$. We know that every clique intersects $A_i \cup B_i$ in an even

number of elements, so it is either disjoint from $A_i \cup B_i$, or intersects both A_i and B_i (because both A_i and B_i are stable sets). Consequently $z^i(T)=0$ ($i=1,\ldots k$ and $\forall\, T\in\mathcal{T}$). $z^1,\ldots z^k$ are linearly independent, because $z^i\equiv t^i \bmod 2$, and $t^1,\ldots t^k$ are independent even over GF(2); Thus a.) is proved.

b.) is an easy consequence of a.): Let $\mathcal{T}$ be a triangle basis over GF(2) and z be an integer vector in the triangle space. Let t be the 0-1vector such that $t\equiv z \pmod 2$. Since $\mathcal{T}$ is also a triangle basis over the rationals, $\mathcal{T}\cup\{z\}$ is dependent, whence $\mathcal{T}\cup\{t\}$ is trivially dependent over GF(2). Since $\mathcal{T}$ is independent over GF(2), t must have nonzero coefficient in the linear dependence of $\mathcal{T}\cup\{t\}$ over GF(2), and the theorem is proved.

Note that we do not know about similar results for perfect graphs with arbitrary chromatic number.

By this theorem, we can restrict ourselves to (mod 2) linear relations for 3-colorable perfect graphs to get more combinatorial type theorems . Let us state for example the following more combinatorial version of Theorem 2.5 for this case:

Let G be a perfect graph with $\omega(G)=3$, and $a,b \in V(G)$. A set $\mathfrak{K}$ of triangles such that a and b are contained in an odd number of triangles of $\mathfrak{K}$, and all other vertices are contained in an even number of triangles of $\mathfrak{K}$, will be called an *(a,b) path of triangles*. Let us define the relation a~b : a~b if and only if there exists an (a,b) path of triangles. If a~b let us say that a and b are *equivalent*.

Theorem 4.3: Let G be a perfect graph, $\omega(G)=3$. The above defined relation is an equivalence relation; $a\neq b\in V(G)$ have the same color in all the 3-colorations of G if and only if they are equivalent; G is uniquely colorable if and only if this equivalence relation has 3 classes.

Proof : The proof is similar to the proof of Theorem 2.5, in fact simpler. Let us just sketch it. First we prove that the defined relation is an equivalence relation:

Let a~b, and suppose a~b and b~c. Let $\mathfrak{K}$ be an (a,b)-path of triangles and $\mathfrak{L}$ a (b,c)-path of triangles. Clearly, $\mathfrak{K} \Delta \mathfrak{L}$ is an (a,c)-path of triangles, which proves transitivity.

The "trivial" if parts of the statements are even shorter here: Let a~b, and let $\mathfrak{K}$ be the 3-cliques which prove this equivalence. Let S_1, S_2, S_3 be a coloration, and

suppose indirectly $a \in S_1$, $b \in S_2$. We obviously have $|\mathfrak{K}| = \Sigma \{\mathfrak{K}(x) : x \in S_i\}$ (i=1,2,3). For i=1,2 this sum is odd, because all terms of it but one ($\mathfrak{K}(a)$ and $\mathfrak{K}(b)$ respectively) are even; on the other hand for i=3 the sum is even, because all terms of it are even, a contradiction.

The essential only if part can be proved similarly to the corresponding part of Theorem 2.5, arguing over GF(2) instead of the rational numbers.

Actually the following strengthening of Theorem 4.3 can be proved:

THEOREM 4.4 : Conjecture 1 is valid for $\omega = 3$.

Here, we omit the proof which is difficult and will be published in a forthcoming paper. Note only, that one of the main difficulty is that contraction by forcing rule does not preserve perfectness.

We saw that Theorem 4.1, which is an easy consequence of Theorem 4.4 immediatly implies the Perfect Graph Conjecture for diamond-free graphs. Another consequence of Theorem 4.4: Tucker's (1984) result about the perfect graph conjecture for K_4-free graphs. (See also Tucker (1987a,b).)

We think that Theorem 4.4 might have further corollaries, that should be exploited in the futur.

5. Coloring algorithms

In this section we shall deduce some algorithmic consequences of the results developed in the previous sections. The general coloration algorithm we suggest below has $O(n^{\omega+1})$ running time, which might be interesting if ω is small. Unfortunately we are unable to find a polynomial time combinatorial algorithm for arbitrary perfect graphs, even if the clique basis is given.

Recall that it is also possible to use Gaussian elimination in order to get a polynomial algorithm for minimum coloring, if the chromatic number is fixed, see Fulkerson(1971). However, the complexity of such an algorithm is much bigger.

The initial idea of the coloring algorithm presented below is the following statement:

Theorem 5.1: Let G be a perfect graph with $\omega = \omega(G)$. There exists an ordering of the vertices of G: $x_1, \dots, x_n$, such that $\{x_{n-\omega+1}, x_{n-\omega}, \dots, x_n\}$ is an ω-clique, and for $1 \le i \le n-\omega+1$: $0 \le r(G_i) - r(G_{i+1}) \le \omega - 1$, where G_i is the graph induced by $\{x_i, x_{i+1}, \dots, x_n\}$, $(i=1,\dots,n-\omega)$.

Proof: We proceed by induction on n. If G contains only one ω-clique, the proposi- tion is clear. Otherwise the statement we have to prove is clearly the following: there exists $x_1 \in V(G)$ such that $0 \le r(G) - r(G-x_1) \le \omega - 1$. But this is just Corollary 2.2 b.) .

Let us call a clique-base $\mathfrak{B}$ *normal* with the order $x_1, \dots, x_n$ of the vertices, if:
a.) $\{x_{n-\omega+1}, x_{n-\omega+2} \dots, x_n\} \in \mathfrak{B}$
b.) for $1 \le i \le n-\omega+1$: $d_i = r(G_i) - r(G_{i+1}) \le \omega - 1$, where d_i is the number of cliques $B \in \mathfrak{B}$, $B \subseteq \{x_i, \dots, x_n\}$containing x_i $(i=1,\dots,n-\omega)$.
(Equivalently: $\mathfrak{B}_i := \{B \in \mathfrak{B} : B \subseteq \{x_i, \dots, x_n\}\}$ is a triangle basis of G_i).

Theorem 5.2: There exists a normal clique basis.

Proof: Take an order ensured by Theorem 5.1, and define $\mathfrak{B}$ recursively:
$\mathfrak{B}_{n-\omega+1} := \{ \{x_{n-\omega+1}, x_{n-\omega}, \dots, x_n\} \}$;
$\mathfrak{B}_i :=$clique basis of the graph induced by $\{x_i, x_{i+1}, \dots, x_n\}$ containing $\mathfrak{B}_{i+1}$ $(i=n-\omega,\dots,1)$
(Clearly, the clique basis $\mathfrak{B}_{i+1}$ of G_{i+1} can be completed to a clique basis $\mathfrak{B}_i$ of G_i.)
If $B \in \mathfrak{B}_i \setminus \mathfrak{B}_{i+1}$, then $x_i \in B$, because if not, then $B \subseteq \{x_{i+1}, x_{i+2}, \dots, x_n\}$, in contradiction with the maximality of $\mathfrak{B}_{i+1}$. Thus $d_i = |\mathfrak{B}_i \setminus \mathfrak{B}_{i+1}| = r(G_i) - r(G_{i+1})$. Q.E.D.

For $\omega=3$: There exists a triangle basis $\mathfrak{B}$ such that $\{x_{n-2}, x_{n-1}, x_n\} \in \mathfrak{B}$, and $\{x_i, x_{i+1}, \dots, x_n\}$ contains 0, 1 or 2 more elements of $\mathfrak{B}$ than $\{x_{i+1}, x_{i+2}, \dots, x_n\}$ depending on whether $r(G_i) - r(G_{i+1})$ is 0, 1 or 2; these new elements contain x_i.

We shall now see a new relation between linear algebra and colorations, a relation that can be used algorithmically. Here we shall restrict ourselves to the case $\omega=3$ in order to show the ideas on a simple special case.

Theorem 5.3: Let G be perfect and $\omega=3$. Let $\mathfrak{B}$ be a normal triangle basis with the order $\{x_1,\ldots, x_n\}$, and use the notation above. Suppose that the graph G_{i+1} has already been colored, and denote by $H_{x,y}$ the graph induced by the vertices of color x and y.

a.)If $r(G_i) - r(G_{i+1})=0$, then for arbitrary two colors a and b, in every component of $H_{a,b}$, all the neighbors of x_i have the same color.

b.)If $r(G_i) - r(G_{i+1})=1$, then for all triangles T of G_i containing x_i, $T\backslash x_i$ is colored with the same two colors, say a and b. Moreover in every component of $H_{c,b}$ and $H_{c,a}$ all the neighbors of x_i have the same color.

c.)If $r(G_i) - r(G_{i+1})=2$, then there exists a color c, such that all triangles of G_i containing x_i have a vertex of color c. Moreover, in every component of $H_{a,b}$, all the neighbors of x_i have the same color.

Proof: First note the following:

(*) Suppose that x_i has two neighbors y and z of different colors in the same component of a graph H induced by their two colors, and let P be a minimal path in H with end-nodes y and z. Then *there exists a triangle consisting of x_i* and *two neighboring vertices of P.*

(Indeed, PUx_i induces an odd cycle and cannot be bipartite since G is perfect.)

Now, if a.) were not true, then by (*) we would have a triangle through x_i, a contradiction.

To prove b.), suppose that there exist triangles T_1, T_2 of G_i, where $T_1\backslash x_i$ is colored with colors "a" and "b", and $T_2\backslash x_i$ is colored with colors "b" and "c". We show that $\mathfrak{B}_{i+1}U\{T_1, T_2\}$ is linearly independent, in contradiction with $r(G_i) - r(G_{i+1})=1$. (Recall that $\mathfrak{B}_j:=\{B\in\mathfrak{B}: B\subseteq\{x_j, \ldots, x_n\}\}$ for $j=1,\ldots,n$.)

$\mathfrak{B}_{i+1}U\{T_1\}$ is linearly independent because of $x_i\in T_1$. *Define now $v(x):=1$ if x has color "a",or "b", and $v(x):=0$ for all other vertices of V(G). Clearly, v(T) is even for all triangles in $\mathfrak{B}_{i+1}U\{T_1\}$, and $v(T_2)$=-is odd, proving the linear independence of T_2 of all the other triangles of $\mathfrak{B}_{i+1}U\{T_1\}$* over GF(2) as well. This contradiction proves that all triangles of G_i containing x_i are colored with the same two colors, a and b say.

The second sentence of b.) follows now immediately from (*): if this second sentence were not true, by (*) there would exist a triangle of G_i containing x_i and having a vertex of color c, in contradiction with the already proven first sentence. Thus b.) is proved.

Let us prove c.) now. Again, the second sentence follows from the first one: using (*) the indirect assumption would imply the existence of a triangle T in G_i containing x_i, where $T \setminus x_i$ is colored with the colors "a" and "b". But this is in contradiction with the first sentence.

To prove the first sentence, suppose indirectly that $T_{a,b}$, $T_{b,c}$, $T_{a,c}$ are triangles of G_i containing x_i , whose two other vertices are colored with the colors shown in the indices. According to the proof of b.), $\mathfrak{B}_{i+1} \cup \{T_{a,b}, T_{b,c}\}$ is linearly independent over GF(2).

Let $v(x) := 1$ if x is of color a or c or if $x = x_i$. Clearly, $v(T) \equiv 0 \bmod 2$ if $T \in \mathfrak{B}_{i+1}$, and also if $T = T_{a,b}$ or $T = T_{b,c}$. On the other hand, $v(T_{a,c}) \equiv 1 \bmod 2$. Hence $T_{a,c}$ does not depend linearly from $\mathfrak{B}_{i+1} \cup \{T_{a,b}, T_{b,c}\}$ over GF(2). Equivalently, $\mathfrak{B}_{i+1} \cup \{T_{a,b}, T_{b,c}, T_{a,c}\}$ is linearly independent, in contradiction with $r(G_i) - r(G_{i+1}) = 2$.

Note how this Theorem implies an algorithm: in a.), by interchanging the two colors in some components of the graphs induced by 2 color classes, all neighbors of x_i will have the same color. In b.) and c.) similarly, after maybe interchanging colors in some components of $H_{x,y}$, x_i will have neighbors of two different colors only. The precise details of the algorithm are omitted here.

Note also that in the proof of c.) the equivalence of the independence over the rationals and over GF(2) played again an important role (Theorem 4.2). Since we do not know this for $\omega \geq 4$, the algorithm for general ω will be less efficient . However, Theorem 5.3 can be straightforwardly generalized to arbitrary ω and the complexity of the corresponding algorithm is $O(n^{\omega+1})$.

Acknowledgment: We are indebted to Gabor Bacso, Michel Burlet, Myriam Preissmann and Péter E. Soltész for the many very useful discussions on the topic of the paper and for helping us with various counterexamples.

References:

V. Chvàtal (1975) On certain polytopes associated with graphs, Journal of Combinatorial Theory (B), **18**, pp 138-154

V. Chvàtal (1984), An equivalent version of the Strong Perfect Graph Conjecture, in Annals of Discrete Mathematics, **21**, "Topics on Perfect Graphs", Edited by C. Berge and V. Chvàtal, North Holland

J. Fonlupt, A. Zemirline (1988) A polynomial recognition algorithm for perfect K_4\e-free graphs, submitted to Journal of Combinatorial Theory (B)

D.R. Fulkerson (1970) The perfect graph conjecture and pluperfect graph theorem, in:Proceedings of the Second Chapel Hill Conference on Combinatorial Mathematics and its applications, R.C. Bose et al. eds., pp171-175

D.R. Fulkerson (1971) Blocking and antiblocking pairs of polyhedra, Mathematical Programming, **1**, 168-194

R. Giles, L.E. Trotter, A. Tucker (1984), The Strong Perfect Graph Theorem for a class of partitionable graphs, in Annals of Discrete Mathematics, **21**, "Topics on Perfect Graphs", Edited by C. Berge and V. Chvàtal, North Holland

L. Lovàsz (1972) Normal hypergraphs and the perfect graph conjecture, Discrete Mathematics, **2**, pp 253-267

S.E. Markossian, G.S. Gasparian (1988), O Gipoteze Berge-a (in Russian), preprint

M. Padberg (1974), Perfect 0-1 Matrices, Mathematical Programming, **6**, 180-196

K.R. Parthasarathy, G. Ravindra (1979) , The validity of the strong perfect graph conjecture for K_4-e-free graphs, J. Comb. Theory, Ser B, **26**, 98-100

A. Schrijver (1986), Theory of linear and integer programming, (Wiley)

A. Tucker (1983), Uniquely Colorable Perfect Graphs, Discrete Mathematics, **44**, 187-194

A. Tucker(1984), The validity of the Perfect Graph Conjecture for K_4-free graphs, in Annals of Discrete Mathematics, **21**, "Topics on Perfect Graphs", Edited by C. Berge and V. Chvatal, North Holland

A. Tucker (1987a) Coloring Perfect K_4-e-Free Graphs, Journal of Comb Theory, B **42**, 313-318

A. Tucker (1987b), A Reduction Procedure for Coloring Perfect K_4-Free Graphs, Journal of Comb Theory, B **43**, 173-186

CONSERVATIVE WEIGHTINGS AND EAR-DECOMPOSITIONS OF GRAPHS

by

András Frank*)

ABSTRACT

An edge-weighting w of a graph $G = (V, E)$ is called conservative if there is no circuit of negative total weight. We prove that the minimum of $w(E)$ over all conservative ± 1 weightings is equal to the maximum number of odd ears in an ear-decomposition of G. As a corollary, we have $\mu = 1/2(\varphi + |V| - 1)$ where μ is the biggest cardinality of a minimum T-join over all even subsets T of V and φ is the minimum number of edges the contraction of which makes the graph factor-critical. The investigation of μ was suggested by Solé and Zaslavsky who observed that μ has another meaning in coding theory: $\mu(G)$ is the covering radius of the cycle code of G. The proof of the main theorem gives rise to a polynomial-time algorithm to construct the optima in question.

I. INTRODUCTION

This paper is concerned with a problem related to matchings, T-joins and ear-decompositions of a graph. For a general account on the topic we refer to the excellent book of Lovász and Plummer [9]. The topic is also related to coding theory. See the book of MacWilliams and Sloane [11].

Let $G = (V, E)$ be a finite undirected graph. By an *ear-decomposition* of G we mean a sequence $G_0, G_1, \ldots, G_t = G$ of subgraphs of G where G_0 consists of one node and no edge, and each G_i arises from G_{i-1} by adding a path P_i for which the two (not-necessarily distinct) end-nodes belong to G_{i-1} while the inner nodes of P_i do not. The paths P_i are called *ears*. Note that P_i may consist of a single edge and such an ear is called *trivial*. Sometimes we say the set $\mathcal{P} = \{P_1, P_2, \ldots, P_t\}$ of paths to be an ear-decomposition. An ear is *odd* if its length is odd. An ear-decomposition is called *odd* if each of its ear is odd.

It is well-known that G has an ear-decomposition if and only if G is 2-edge-connected

*) Institute for Operations Research, University of Bonn, Nassestr. 2, 5300 Bonn 1, West Germany. On leave from Department of Computer Science, Eötvös University, Budapest

and **we will assume that each graph in this paper is 2-edge-connected.** Actually, an ear-decomposition of any 2-edge-connected subgraph of G can be continued to become an ear-decomposition of G. In particular, any circuit of G can be chosen to be the first member of an ear-decomposition.

By induction it is clear that the number t of ears of an ear-decomposition is independent of the decomposition and equals $|E| - |V| + 1$.

There are various known results on ear-decompositions. [9]. A basic one is due to L.Lovász [7]. To formulate it we recall that a graph is called *factor-critical* if $G - v$ has a perfect matching for every node v of G. (Factor-critical graphs are sometimes called hypomatchable.)

THEOREM 1.1 [7] *A graph is factor-critical if and only if it possesses an odd ear-decomposition. Furthermore, for any edge e of a factor-critical graph G there is an odd ear-decomposition of G such that the first ear uses e.*

The second statement is not explicitely stated in Lovász' paper but it immediately follows from his proof. (It is not true that any odd circuit can be the first ear of an odd ear-decomposition.) One of our purposes is to answer the following question.

What is the minimum number $\varphi = \varphi(G)$ of even ears in an ear-decomposition of G?

Lovász' theorem can be formulated this way: given a graph G, $\varphi(G) = 0$ if and only if G is factor-critical.

We call en ear-decomposition *optimal* if the number of even ears is φ.

There are some results in the literature, besides Lovász theorem, concerning this question. For example, Hetyei [6], [9: Theorem 5.4.1] proved that for matching-covered graphs $G\varphi(G) = 1$. Another result (see Theorem 1.3 below) asserts that for a bipartite graph $G\varphi(G) = 1$ if and only if G is elementary.

We are going to prove a good characterization for general graphs having $\varphi(G) = 1$. (Corollary 2.6) A more general purpose of the paper is to provide a min-max formula for φ. Namely, we prove that $\varphi(G) = 2\mu(G) - |V| + 1$.

Here $\mu(G)$ is a parameter that was introduced by Solé and Zaslavsky [15], as follows. A ± 1 weighting $w : E \to 1, -1$ of G is called *conservative* if there is no circuit of negative total weight. Let $\mu = \mu(G)$ denote the maximum number of negative edges in a conservative weighting. We call a conservative weighting realizing μ *optimal.*

Originally, Solé and Zaslavsky investigated μ in a different context. They observed that μ is equal to the covering radius of the cycle code of G. The investigation of covering radius was originated by MacWilliams and Sloane in their book [11: page 173, Research

Problem (6.1)]. (We do not introduce these concepts here. Interested readers should contact [15]). Via this observation of Solé and Zaslavsky, the present paper can be considered as a complete answer to the above research problem for the class of cycle codes. (A binary code can be identified with a binary subspace. A cycle code [15] is one identified with the cycle space of an undirected graph.)

For a subset F of edges we use the notation $w(F) := \sum(w(e) : e \in F)$.

Conservative ± 1 weightings are strongly related to T-joins. Let T be a subset of V with even cardinality. A subset F of edges is called a *T-join* if a node is incident to an odd number of edges from F precisely if v belongs to T. Note that any subset F of edges is a T_F-join where T_F is defined to be the subset of nodes with an odd number of incident edges from F.

For a ± 1 weighting w let F_w denote the set of negative edges. For a subset F of edges let w_F be defined by $w_F(e) := -1$ if $e \in F$ and $w_F(e) := +1$ if $e \in E - F$. We call a subset F of edges a *join* if w_F is conservative.

There is an extensive literature on $T-$joins and T-cuts [2, 3, 8, 12, 13, 14]. Here we mention only an easy but basic result, due to A. Sebö .

PROPOSITION 1.2 [13] *Let w be a conservative weighting of a graph $G = (V, E)$ and T a subset of nodes of even cardinality. A T-join F is of minimum w-weight if and only if w' is conservative where $w'(e) := -w(e)$ if $e \in F$ and $w'(e) := w(e)$ if $e \in E - F$. In particular, if F is either a circuit of zero w-weight (case of $|T| = 0$) or a path of minimum w-weight, connecting two specified nodes of G (case of $|T| = 2$), then changing the sign of w along F results in a conservative weighting.*

Specializing Proposition 1.2 to $w \equiv 1$ one obtains a result of Mei-gu Guan [5] that serves as a bridge between minimum cardinality T-joins and conservative weightings: a T-join F is of minimum cardinality if and only if there is no circuit of negative total w_F-weight.

This means that a minimum cardinality T-join can be determined if one is able to decide whether there is a negative circuit with respect to a ± 1 weighting. The other direction is true, as well. That is, if one wants to decide whether a given ± 1 weighting w is conservative, one can do it by determining the minimum cardinality of a T_F-join where F is the set of negative edges.

From these considerations it is clear that μ can be interpreted as the biggest cardinality of a minimum T-join over all even subsets T of nodes. Since a subset F of edges is a join if and only if F is a minimum T_F-join, we have yet another meaning of μ: it is the maximum cardinality of a join.

To complete this section we introduce some further notation, notions, and theorems

from matching theory (see [9]). For a subset X of nodes of a graph G let $\Gamma(X) := \{v \in V - X$: there is an edge $uv \in E$ with $u \in X\}$ and let $q(X)$ be the number of odd components in $G - X$.

By a *matching* M we mean a subgraph of G with no two adjacent edges. A *perfect matching* is one covering all the nodes of G. A graph G having a perfect matching is called *perfectly matchable* (or, in short, *matchable*). Let G be perfectly matchable. G is *critical* if $G - \{u, v\}$ has a perfect matching for any two distinct nodes u, v. G is *matching covered* if every edge belongs to a perfect matching. G is *elementary* if the union of perfect matchings forms a connected spanning subgraph of G. We call a matching M of a graph a *near matching* if M leaves precisely one node exposed.

We have already introduced the concept of factor-critical graphs along with Lovász' theorem. Let us cite here three other fundamental theorems (besides Tutte's theorem).

THEOREM 1.3 [9: *Theorems 4.1.1 and 4.1.6] For a bipartite graph* $G = (A, B; E)$ *with* $|A| = |B|$ *the following are equivalent.*

a. *G is elementary,*

b. *G is matching covered,*

c. *G has an ear-decomposition such that the first ear is even while all other ears are odd.*

d. $|\Gamma(X)| > |X|$ *For every proper subset* $\emptyset \neq X \subset A$.

e. *For a perfect matching M the w_M-distance of any two nodes is non-positive.*

□

In particular, if G consists of parallel edges between two nodes, then G is elementary biparite.

Let us recall Tutte's theorem: A graph has a perfect matching if and only if $q(X) \leq |X|$ for each subset X of nodes. The deficiency $def(G)$ of a graph G is defined to be $max(q(X) - |X| : X \subseteq V)$. A set X is with $q(X) - |X| = def(G)$ is called a *barrier* of G. Obviously any matching of G leaves at least $def(G)$ nodes exposed and the Berge-Tutte formula, a slight generalization of Tutte's theorem, asserts that any matching of maximum cardinality leaves precisely $def(G)$ nodes exposed.

For a matchable graph G we call a barrier X a *strong barrier* if $G - X$ includes $|X|$ odd components and no even components, each odd component is factor-critical, and shrinking the odd components and deleting the edges induced by X results in an elementary bipartite graph. We call a perfectly matchable graph *half-elementary* if it has a strong barrier. The name is justified by the following theorem.

THEOREM 1.4 [9: *Theorem 5.2.2] Let G be an elementary graph. Define two nodes x, y to be in relation if either $x = y$ or $G - \{x, y\}$ is* not *matchable. Then this relation is*

an equivalence relation and each equivalence class is a strong barrier.

□

Let $G = (V, E)$ be a graph and $H = (U, A)$ a half-elementary graph with a strong barrier X. Assume that $V \cap U = \emptyset$. The following operation is fundamental in our investigations. By *implanting H along X at a node v of G* we mean the following operation: delete v from G and for each edge uv of G introduce an edge from u to an arbitrary node of X.

Suppose that G has no perfect matching. Let $D(G)$ denote the set of nodes exposed by at least one maximum matching and let $A(G) := \Gamma(D(G))$. In the literature the following fundamental result is called the Gallai-Edmonds structure theorem.

THEOREM 1.5 [1,4] *$A(G)$ is a barrier and $D(G)$ is the union of odd components of $G - A(G)$. $A(G)$ can be described as the unique barrier X for which the union of odd components of $G - X$ is minimum. Moreover, the odd components in $D(G)$ are factor-critical, and for any non-empty subset X of $A(G)$ the number of odd components in $D(G)$ having a neighbour in X is bigger than $|X|$.*

□

II. MAIN RESULTS

The main result of this paper provides a relationship between μ and φ.

THEOREM 2.1

$$\varphi(G) = 2\mu(G) - |V| + 1. \tag{2.1}$$

The idea behind the proof is as follows. First we prove the theorem for factor-critical graphs. Then it will be shown that every graph can be built up from a factor-critical graph in a certain way. Finally, by analysing the behaviour of the values φ and μ during one step of the construction, we obtain (2.1).

Since the intermediate steps of the proof provide some insight into the structure of maximum joins, we formulate them as separate theorems.

THEOREM 2.2 *Every graph G can be built up from a factor-critical graph by consecutively implanting half-elementary graphs at arbitrary nodes.*

THEOREM 2.3

(a) If G is a factor-critical graph, then $\varphi(G) = 0$ and any near-perfect matching of G is

a maximum join, that is, $\mu(G) = (|V| - 1)/2$.

(b) If H is a half-elementary graph, then any perfect matching of H is a maximum join, that is, $\mu(H) = |V(H)|/2$. *Furthermore,* $\varphi(H) = 1$ *and there is an optimal ear-decomposition starting with an even ear.*

THEOREM 2.4 *Suppose that $G = (V, E)$ arises from a graph $G' = (V', E')$ by implanting a half-elementary graph H. Then* $\mu(G) = \mu(G') + |V(H)|/2$ *and* $\varphi(G) = \varphi(G') + 1$. *There is a maximum join of G which is the union of a maximum join of G' and a perfect matching of H. There is an optimal ear-decomposition of G that starts with an ear-decomposition of H.*

The proofs will appear in Section 3. Here we list some consequences but before doing so let us mention that φ can be interpreted as the minimum number of edges the contraction of which makes the graph factor-critical. (Proposition 3.3).

COROLLARY 2.5 *The maximum number of odd ears in an ear-decomposition of $G = (V, E)$ is equal to the minimum of $w(E)$ over all conservative ± 1 weightings w of G.*

Proof. The maximum in question is $|E| - |V| + 1 - \varphi(G)$ since the total rs number of ears in any ear-decomposition is $|E| - |V| + 1$. On the other hand the minimum can be expressed as $|E| - 2\mu(G)$. By these two observations (2.1) implies the statement.

□

The theorems above imply that if $\varphi(G) \geq 1$, then there is an optimal ear-decomposition starting with an even circuit. If $\varphi(G) = 1$, then the first ear is even and the other ears are odd. Obviously such a graph has a perfect matching.

COROLLARY 2.6 *Let M be a perfect matching of a graph $G = (V, E)$. The following are equivalent:*

a, $\varphi(G) = 1$,

b, M is a maximum join,

c, the w_M-distance of any two nodes is non-positive.

d, There are no two disjoint non-empty subsets A, B of nodes so that $G - (A \cup B)$ contains $|A| + |B|$ odd components (i.e. $A \cup B$ is a barrier) among which $|A|$ components are connected only to A and the other $|B|$ components are connected only to B.

Proof. a → b. By (2.1) we have $\mu(G) = |V|/2$, that is, M is a maximum join.

b→c. Immediate from Proposition 1.2.

c→d. If there were two subsets A, B with the given properties, then the w_M distance of any node of A and any node of B would be positive.

d→a. By Theorem 2.2 G arises from a certain G' by implanting at a node v of G' a half-elementary graph along a strong barrier A. We claim that G' is factor-critical. Indeed, if G' is not factor-critical, then $B := A(G')$ is non-empty. Now by Theorem 1.5 $v \notin B$ as M restricted to G' is a maximum matching of G' avoiding v. It follows that A and B satisfy the properties in d.

Therefore G arises from a factor-critical graph by implanting one half-elementary graph and hence by Theorem 2.4 we have $\varphi(G) = 1$.

□

REMARK for readers interested in coding theory. From (2.1) it follows that the covering radius $cr(G)$ of the cycle space of a connected graph $G = (V, E)$ is always at least $\lfloor n/2 \rfloor$ where $n = |V|$. For odd n $cr(G)$ is precisely $\lfloor n/2 \rfloor$ if and only if G is factor-critical. For even n Corollary 2.6 describes the graphs for which $cr(G) = \lfloor n/2 \rfloor$. The next result may also have some interest for coding theorists.

COROLLARY 2.7 *If $G = (V, E)$ is a subgraph of $G' = (V, E')$, then $\mu(G) \geq \mu(G')$.*

Proof. Immediate from Theorem 2.1.

□

III. PROOFS

Proof of Theorem 2.2. There is nothing to prove if G is factor-critical. Suppose it is not. Then we are going to show that G includes a half-elementary graph H as an induced subgraph so that a strong barrier X of H contains all such nodes of H that have neighbours in $V - V(H)$. The existence of such an H shows that G can be constructed from G' by implanting H where G' denotes the graph obtained from G by contracting H into one node. This way one can inductively go back to a factor-critical graph.

We claim that G contains a non-empty subset Z of nodes so that $G - Z$ contains at least $|Z|$ factor-critical components. Indeed, if G has no perfect matching, then $Z = A(G)$ will do. If G is perfectly matchable, then $Z = A(G - v) + v$ satisfies the requirement for any node v.

Among the $|Z|$ factor-critical components of $G - Z$ choose a minimum number j so that they have at most j neighbours in Z. This number of neighbours is precisely j. Let us denote the union of the j components by Y and $X := \Gamma(Y)$. Now it is easily seen that the subgraph H of G induced by $X \cup Y$ is half-elementary with strong barrier X and it satisfies the requirements.

□

PROPOSITION 3.1 *Let $G = (V, E)$ be a graph obtained from $G' = (V', E')$ by adding an ear P of length p. Then $\mu(G') \geq \mu(G) - \lfloor p/2 \rfloor$.*

Proof. Let u and v be the two (not necessarily distinct) w w end-nodes of P. Let be an optimal ± 1 weighting of G. Let $p^-(p^+)$ denote the number of negative (positive) edges of P and let w' denote the restriction of w to E'. First suppose that $p^- \leq p^+$. Then $p^- \leq \lfloor p/2 \rfloor$ and we have $\mu(G') \geq \mu G) - p^- \geq \mu(G) - \lfloor p/2 \rfloor$, as required. Second, suppose that $p^- > p^+$, that is, $w(P) < 0$. Then every path in G' connecting u and v has w'-weight at least $-w(P)$, a positive number. Proposition 1.2 shows that changing the sign of w' along a path of minimum w'-weight provides a conservative weightings w" of G' and the number of edges with negative w"-weights is $\mu(G) - p^- - w(P)$. Therefore $\mu(G') \geq \mu(G) - p^- - w(P) = \mu(G) - p^+ \geq \mu(G) - \lfloor p/2 \rfloor$, as required.

□

PROPOSITION 3.2

$$\varphi(G) \geq 2\mu(G) - |V| + 1. \tag{3.1}$$

Proof. By induction on the number of nodes. Let P be the last ear of an ear-decomposition of G with $\varphi(G)$ even ears. Let ϵ be 0 if P is odd and 1 if P is even. Using Proposition 3.1 we have $\varphi(G) = \varphi(G') + \epsilon \geq 2\mu(G') - |V'| + \epsilon + 1 = 2(\mu(G) - \lfloor p/2 \rfloor) - |V'| + \epsilon + 1 = 2\mu(G) - (p - 1 + |V'|) + 1 = 2\mu(G) - |V| - 1$.

□

Proof of Theorem 2.3.

(a) If G is factor-critical, then $\varphi(G) = 0$ and by Proposition 3.2 $\mu(G) \leq (|V| - 1)/2$. Moreover a near-perfect matching M of G is a join and hence $\mu(G) = (|V| - 1)/2$, that is M is an optimal join.

(b) Let X be a strong barrier of H. Since H has an even number of nodes, $\varphi(H) \geq 1$. To see the equality we can suppose that X induces no edge since deleting these edges leaves H half-elementary. We use induction on the number of nodes. If every component of $H - X$ is a singleton, then H is elementary bipartite and, by Theorem 1.3, $\varphi(H) = 1$. Moreover the first ear in an ear-decomposition of H is even.

Now let K be a component of $H - X$ with $|K| > 1$. Let H' denote the graph obtained from H by shrinking K into a new node v_K. H' is half elementary as X is a strong barrier of it. By induction, H' has an ear-decomposition $\mathcal{P}'$ so that the first ear is even while all the other ears are odd. Let P be the first ear in $\mathcal{P}'$ that contains v_k. Suppose that x_1v_1 and x_2v_2 are edges in H corresponding to the two edges of P incident to $v_K(x_1, x_2 \in X$

and $v_1, v_2 \in K$). Introduce a new edge $e = v_1v_2$ in K. By Theorem 1.1 there is an odd ear-decomposition $\mathcal{P}_K$ of K so that the first ear P_0 contains e. P_0 is an odd circuit and hence $P_0 - e$ is an even path in K connecting v_1 and v_2. Define P' by replacing the subpath (x_1v_K, v_Kx_2) by $(x_1v_1, P_0 - e, v_2x_2)$. Obviously the length of P' has the same parity than that of P.

Now we can define an ear-decomposition of H using one even ear as follows. In $\mathcal{P}$ replace P by P' and insert $\mathcal{P}_K - \{P_0\}$ right after P'. Thus $\varphi(H) = 1$.

By Proposition 3.2 $\mu(H) \leq |V(H)|/2$. Here we have equality as any perfect matching is a join.

□

We need the following interpretation of φ.

PROPOSITION 3.3 *$\varphi = \varphi(G)$ is the minimum number of edges the contraction of which makes G factor-critical.*

Proof. Let us consider an ear-decomposition $\mathcal{P}$ of G with φ even ears. Choose one edge from each even ear. By contracting these φ edges $\mathcal{P}$ transforms into an odd ear-decomposition of the contracted graph G'. By Theorem 1.1 G' is factor-critical, showing that not more than $\mathcal{P}$ edges are required to make G factor-critical.

Conversely, assume that contracting k edges of G makes G factor-critical. In order to show that $\varphi(G) \leq k$ we use induction on k. The inequality is true for $k = 0$ by Theorem 1.1. Let $e = uv$ one of the k contracted edges and let G' denote the graph obtained from G by contracting e. Denote by v_e the contracted node. Then G' can be made factor-critical by contracting $k - 1$ edges and hence, by induction, G' has an ear-decomposition $\mathcal{P}'$ using at most $k - 1$ even ears. Let P' be the first ear in $\mathcal{P}'$ containing the v_e.

P' corresponds to a path P in G. Let $\mathcal{P} := \mathcal{P}' - \{P'\} \cup \{P\}$. e is either an element of P or one of u and v, say u, belongs to P while v does not. In the first case $\mathcal{P}$ is an ear-decomposition of G in which the number of even ears is at most on e bigger than that in $\mathcal{P}'$. Therefore $\varphi(G) \leq k$. In the second case $\mathcal{P}$ is an ear-decomposition of $G - v$ using $k-1$ even ears. Extending $\mathcal{P}$ by an even ear (uv, vt), where vt is an edge of G distinct from uv, we obtain an ear-decomposition of G using at most k even ears. Therefore $\varphi(G) \leq k$ holds in this case, as well.

□

PROPOSITION 3.4 *For any edge f of G there is an optimal ear-decomposition of G such that the first ear uses f.*

Proof. Let us choose an ear-decomposition of G that uses $\varphi(G)$ even ears. Choose an edge e from one of these even ears that is distinct from f. Let G' denote the graph obtained

from G by contracting e. By the preceding proposition $\varphi(G') = \varphi(G) - 1$. By induction G' has an optimal ear-decomposition such that the first ear uses f. By the second part of the proof of the preceding proposition this ear-decomposition defines an optimal ear-decomposition of G.

□

For the next two propositions suppose that G arises from G' by implanting a half-elementary graph H along a strong barrier X of H at a node v_X of G'.

PROPOSITION 3.5

$$\varphi(G) \leq \varphi(G') + 1. \tag{3.2}$$

Proof. By Theorem 2.1 H has an ear-decomposition $\mathcal{P}_1$ using one even ear. By Proposition 3.4 G' has an optimal ear-decomposition $\mathcal{P}'$ such that the first ear contains v_X. Any ear P' in $\mathcal{P}'$ for which both end-nodes are v_X corresponds to a path P" by of G having both end-nodes in X. Let $\mathcal{P}$" be obtained from $\mathcal{P}'$ replacing all such P' by P". Then $\mathcal{P} := (\mathcal{P}_1, \mathcal{P}")$ is an ear-decomposition of G using $\varphi(G') + 1$ even ears. Therefore $\varphi(G) \leq \varphi(G') + 1$.

□

PROPOSITION 3.6

$$\mu(G) \geq \mu(G') + |V(H)|/2. \tag{3.3}$$

Let F' be an optimal join of G' and let F denote the subset of edges of G corresponding to F'. Let M be any perfect matching of H. We claim that $M \cup F$ is a join of G. Indeed let C be any circuit C of of G. If C is disjoint from X then it is either a circuit of H or corresponds to a circuit of G' avoiding v_X. In both cases we have $w_{F \cup M}(C) \geq 0$ as M is a join of H and F' is a join of G'. If C intersects X, then C partitions into edge-disjoint paths so that the end s of these paths belong to X while the inner nodes (if any) do not. Therefore the $w_{F \cup M}$-weight of each such path is non-negative and thus $w_{F \cup M}(C) \geq 0$.

□

Proof of Theorem 2.1. If G is factor-critical, then $\varphi(G) = 0$ and (2.1) is equivalent to $\mu(G) = (|V| - 1)/2$. That was already proved in Theorem 2.3. If G is not factor-critical, then, by Theorem 2.2, G arises from a certain G' by implanting a half-elementary graph H. By induction we assume that (2.1) holds for G', that is $\varphi(G') = 2\mu(G') - |V'| + 1$.

Putting together this, (3.1), (3.2), and (3.3) we get

$$\varphi(G) \leq \varphi(G') + 1 = 2\mu(G') - |V'| + 2 \leq$$
$$2\mu(G) - |V(H)| + |V'| + 2 = 2\mu(G) - |V| + 1 \leq \varphi(G). \tag{3.4}$$

Hence equality holds everywhere from which (2.1) follows.

□

Proof of Theorem 2.4. By (3.4) we have equality in (3.2) and in (3.2). Therefore the join $M \cup F$ constructed in the proof of Proposition 3.6 is maximum and it has the property required in the theorem. The equality in (3.2) shows that the ear-decomposition $\mathcal{P}$ given in the proof of Proposition 3.5 is optimal and it has the property required in the theorem.

□

IV. ALGORITHMIC ASPECTS

In this last section we make some comments on the algorithmic aspects of determining an optimal join and an optimal ear-decomposition.

A basic subroutine we need is an algorithm to compute a maximum matching of a graph G along with the unique Gallai-Edmonds barrier $A(G)$. The first such procedure is due to J.Edmonds [1]. The core of Edmonds' algorithm is an augmenting step, we call it Edmonds-augmentation, that starts with a matching and either finds a bigger matching or finds $A(G)$. Note that one Edmonds augmentation can be carried out in $O(|E|)$ steps.

If the graph is factor-critical, then any near-perfect matching is an optimal join and Lovász' proof in [7] provides a good algorithm to compute an odd ear-decomposition. (In a total it needs at most $2|V|$ Edmonds-augmentations).

If the graph is not factor-critical, then we have to be able to construct the decomposition described in Theorem 2.2. From such a decomposition an optimal join can be obtained by putting together a near-perfect matching of the starting factor-critical graph and a perfect matching of each implantend half-elementary graph. The proof of Propositions 3.4 and 3.5 shows how to construct an optimal ear-decomposition.

Let us concentrate on computing one implantation. First we compute a maximum matching of G and the set Z given in the proof of Theorem 2.2. This can be done by at most $|V|/2$ Edmonds-augmentations.

Next, we have to determine algorithmically a minimum number of odd components as is described in the proof. This problem can be formulated as determining a subset

$Y \subseteq B$, in a certain bipartite graph $G_1 = (Z, B; E_1)$ having a perfect matching M_1, with the property that Y has $|Y|$ neighbours in Z and every proper non-empty subset Y' of Y has more than $|Y'|$ neighbours in Z. (In the proof of Theorem 2.2 G_1 arises as follows. Given Z as in the proof, choose a subset M_1 of the existing maximum matching that covers Z. Shrink each odd component hit by M_1 into one node. The set of shrunken nodes is B and the set of edges connecting Z and the shrunken nodes forms E_1).

Associate a directed graph as G_2 on node-set Z with the pair (G_1, M_1), follows. In G_2 let uv $(u, v \in Z)$ be a directed edge if $u'v$ is an edge in where $u'(\in B)$ denotes the node for which $uu' \in M_1$. It easy to see that a strongly connected component C of G_2 with no out-going edges determines the required subgraph of G_1 and C can be computed in $O(E_1)$ time.

Therefore one implantation can be computed by using not more than $|V|/2$ Edmonds augmentations. Maintaining the current matching any subsequent implantation can be computed by using only one more Edmonds-augmentations. Therefore the total number of these augmentations is at most $O(|V|)$. Hence the overall complexity of the algorithm can be bounded by $O(|V||E|)$.

REFERENCES

[1] Edmonds, J. (1965): Paths, trees, and flowers. Canad. J. Math. *17*, 449-467.

[2] Edmonds, J. and Johnson, E. (1973): Matching, Euler tour and the Chinese postman. Mathematical Programming *5*, 88-124.

[3] Frank, A., Sebö, A. and Tardos, E. (1984): Covering directed and odd cuts. Mathematical Programming Studies *22*, 99-112.

[4] Gallai, T. (1963): Neuer Beweis eines Tutte'schen Satze. Magyar Tud. Akad. Mat. Kut. Int. Közl. *8*, 135-139.

[5] Mei-Gu Guan, (1962): Graphic programming using odd or even points. Chinese Mathematics *1*, 273-277.

[6] Hetyei, G. (1964): Rectangular configurations which can be covered by 2*1 rectangles. Pécsi Tan. Föisk. Közl. *8*, 351-367 (in Hungarian).

[7] Lovász, L. (1972): A note on factor-critical graphs. Studia, Sci. Math. Hung. , 279-280.

[8] Lovász, L. (1975): 2-matchings and 2-covers of hypergraphs. Acta Mat. Acad. Hungar. *26*, 433-444.

[9] Lovász, L. and Plummer, M. (1986): Maching Theory. Akadémiai Kiadö Budapest and North-Holland Publishing Company, 1986.

[10] Lucchesi, C.L. and Younger, D.H. (1978): A minimax relation for directed graphs. J. London Math. Soc. (2) *17*, 369-374.

[11] MacWilliams, F.J. and Sloane, N.J.A. (1977): The theory of error correcting codes. North-Holland, 1977.

[12] Sebö, A. (1987): A quick proof of Seymour's theorem on T-joins. Discrete Mathematics *64*, 101-103.

[13] Sebö, A. (1989): Undirected distances and the postman structure of graphs. J.Combinatorial Theory, Ser.B., to appear in 1989.

[14] Seymour, P.D. (1981): On odd cuts and plane multicommodity flows. Proceedings of the London Math. Soc. *42*, 178-192.

[15] Solé, P. and Zaslavsky, Ph. (): Covering radius and maximality of the cycle a graph. manuscript.

A Survey of Some Results in Discrete Optimisation

A.A. Fridman and E.V. Levner
Central Economical and Mathematical Institute
32 Krasikova st. Moscow 117418 USSR

Extended Abstract

The aim of this survey is to summarize some recent results concerning polynomial-time exact and approximate algorithms for discrete optimisation problems which were published in Russian.

1 Reducibility among discrete problems and polynomial algorithms

Many integer programming and combinatorial optimisation problems can be solved efficiently (polynomially) by means of their reduction to other, polynomially solvable, problems, e.g., to network flow problems. In their earlier paper, Fridman, Litvak and Rappoport (1973) have suggested to consider a linear programming problem P to be reducible to a min-cost network flow problem P_1 if certain projections of optimal solutions for P_1 onto some subset appear to be the optimal solutions for P. These authors have developed in 1973-1985 a theory of the so-called *M-matrices* which are a special case of the absolutely unimodular matrices, and proved that a linear programming (LP) problem is polynomially reducible to a min-cost network flow problem if the constraint matrix of the LP problem is an M-matrix.

For network flow problems, algorithms have been known for decades which are capable of solving instances with thousands of nodes and arcs. Beautiful results in developing efficient algorithms and computer programs for those

problems being known to belong, among others, to Adelson, Dinic, Karzanov, Cherkasski, Papernov (1975-1988).

Speaking of reducing integer programming problems (IPP), we would like to mention the reduction of IPP to a triangular (diagonal) form is of the same importance for Integer Programming as the Gauss method for the linear programming problems. Frumkin and Votyakov (1973-1986) have developed polynomial algorithms for solving homogeneous integer systems, for finding the rank of a matrix and reducing integer matrices to the Hermite normal triangular form and the Smith diagonal form. These algorithms are based on a polynomial procedure for solving the Diophantine equation systems, which is called *"the matrix analogue of the Euclid algorithm"*. It is worth noticing that while the original Hermite method for solving the Diophantine equation systems may require very large coefficients of intermediate matrices (which may be of order m^3, m being the maximal absolute value of matrix coefficients and right-hand side coefficients), this negative property is absent in the matrix analogue of the Euclid algorithm.

A good illustration of using the reducing techniques for solving efficiently combinatorial problems, is the so-called *"plant location problem"* which is known to have many equivalent forms: it can be written as a special integer programming problem, as a problem of finding a parametric range of machines, as a problem of minimizing a pseudo-boolean function, as an optimal trajectory problem, as a set covering problem, as a problem of minimizing a set function, etc. There is a monograph and several extended surveys in Russian dealing with the problem (Beresnev, Gimadi and Dementyev, 1978; Grishukhin, 1987; Gimadi, 1988), which contain a number of theoretic and algorithmic results rediscovered later in the West. Notice that the polyhedron of the relaxed plant location problem has a simple structure (see the next section).

Another area where the problem reduction turns out to be very fruitful is *scheduling problems*. For example, a rather extensive family of two-machine Johnson-type scheduling problems (which were originally introduced and studied in papers by Johnson (1954), Bellman (1956), Jackson (1956), Mitten (1959), Nabeshima (1963), Szwarc (1968-1983), Kurisu (1973, 1976), Levner (1973), Tanayev (1975), Rinnooy Kan (1976), Maggu and Das (1977-1982), Yoshida and Hitomi (1979), Sule (1982-1983), Monma and Rinnooy Kan (1983), Bagga and Khurana (1984), has been considered by Levner (1988-1989) who put all the problems of the family in a common network framework

and obtained polynomial algorithms for some other scheduling problems of this kind, not necessarily two-machine ones, by reducing such problems to a basic two-machine Johnson-type scheduling problem. Another example of developing a new effective algorithm for the generalized Kelley-Fulkerson project scheduling problem, which is due to Levner and Nemirovsky (1989).

We conclude this section by surveying some results concerning *reciprocity in discrete optimisation problems.* Recall that the *reciprocal problem* is obtained from a (vector) optimisation problem by exchanging some objective functions for some constraining functions (Aganbegyan and Bagrinovski, 1968; Levner and Vladuc, 1983, 1988; Favati and Pappalardo, 1985). Levner and Vladuc (1988) established necessary and sufficient conditions for an optimal (efficient) solution of a primal vector optimisation problem to be an optimal solution of its reciprocal, and showed that any vector integer programming problem can be reduced to a finite series of the (scalar) integer knapsack problems.

2 Geometric arguments in Discrete Optimization

While geometric ideas and results in constructing general Linear and Convex Programming methods due to Khachian and Nemirovsky (1980-1989) are well known in the West and have became now classical, in the USSR there appeared also a number of less known works, devoted to studying *polyhedra and their properties for special combinatorial problems*, such as the transportation problem (Emelichev, Kovalyov and Kravtsov, 1965-1988; Shvartin, 1978-1988), the plant location problem (Trubin, 1973-1988; Beresnev and Gimadi, 1973-1988; Grishukhin, 1977-1989; Ageev, 1983-1988), scheduling problems (Shevchenko, 1979; Suprunenko, 1973-1983; Tanayev, 1980-1988; Leontyev, 1979-1989), that complement the well-known results in this field obtained, for example, by Balas, Balinski, Edmonds, Grotschel, Padberg and Pulleyblank.

As we have mentioned above, the polyhedron of the linear programming relaxation of the plant location problem is of a special simple structure and, due to this fact, Trubin, Beresnev and Gimadi, Grishukhin, Ageev could find an extensive class of problem's functionals which have an optimal value to

be achieved in integer points (vertices) of the relaxed problem polyhedron. These authors found and investigated two important functional's properties, each guaranteeing the polynomial solvability of the allocation problem, which were called the *"quasiconvexity of the constraint matrix"* and the *"connectivity of the matrix"*. In the first case the problem is proved to be solved by a greedy procedure, and in the second case, by a polynomial dynamic programming algorithm, the complexity of the latter depending only on the problem size.

We also consider in this section some applied combinatorial problems of geometric nature, concerned with the *"best" imbedding of a given body (or a body of a given form) into a polyhedron.* Fridman and Votyakov (1969-1985) have determined the minimal number of parameters sufficient for representing octahedral polyhedra (O.P.), and proved that any O.P. is, in fact, an intersection of two regular tetrahedra; these authors also have found sufficient conditions for effective imbedding various geometric bodies into O.P., the latter problem being very important in the utilization of natural monocrystal resources.

In this section, we also consider interfaces between Discrete Optimisation and Mathematical Analysis, concentrating mainly on the geometric arguments in Steinitz's vector-sum theorem (1913) and Bergstrom's lemma (1931) rediscovered by Kadec in 1953 and used later for solving approximately (with performance guarantees) the well-known Johnson and Ackers-Fridman scheduling problems (Belov, Stolin, Sevastyanov, 1974-1988).

3 Fast approximation algorithms with guaranteed performance

Though it is a well known fact now that to find good (in some sense) approximate solutions for many NP-hard combinatorial problems is a NP-hard problem as well (Livshic, 1968; Nigmatullin, 1975; Sahni and Gonzales, 1976; Levner and Gens, 1979), nevertheless there exist classes of NP-hard combinatorial problems which permit us to find out effectively an *approximate solution with the arbitrary guaranteed accuracy* ε. Historically, the first algorithms of this type were developed (for the knapsack problem) by Babat (1975), Ibarra and Kim (1975), Karp (1975) and Sahni (1975).

Following these lines, a number of authors could find fast ε-approximation algorithms (or, *fully polynomial approximation schemes*) for various classes of scheduling, set partitioning, covering and packing problems, multicriterial and multidimensional knapsack, and other discrete optimisation problems (Sahni, 1976, 1977; Horowitz and Sahni, 1976; Sahni and Gonzales, 1976; Chandra, Hirschberg and Wong, 1976; Gens and Levner, 1977-1988; Ibarra and Kim, 1978; Lawler, 1978, 1979, 1982; Babat, 1978; Kannan and Korte, 1978; Korte and Schrader, 1980; Moran, 1981; Gens, 1981, 1984; Magazine and Oguz, 1981; Karp and Karmarkar, 1982; Levin, 1982; Schrader, 1983; Johnson and Niemi, 1983; Hochbaum and Maase, 1984; Kovalyov, 1984-1988; Kovalyov and Shafransky, 1985-1987; Babat and Vladuc, 1988).

For example, Karmarkar and Karp (1982) devised a fully polynomial approximation scheme for the *one-dimensional bin packing problem*; Lawler (1982) presented such a scheme for *the total tardiness scheduling problem*, Johnson and Niemi (1983) solved ε-approximately *the knapsack problem with tree-like precedences*, Gens and Levner (1988) suggested an algorithm proved to generate in polynomial time an *ε-approximate lattice of Pareto-optimal solution set for the vector knapsack problem*; Babat and Vladut (1988) described how to find in polynomial time ε-approximate, ε-feasible solutions of the *multidimensional knapsack problem.*

We would like to mention here that there is another approach to solbing approximately discrete optimisation problems, closely related to the considered ε-approach, which deals with *fuzzy formulation* of the problems (Lebedev, 1987-1989; Levner and Ptuskin, 1987-1989).

In this section we also survey recent Russian-published results on approximation algorithms guaranteeing "near-optimal" solutions for various machine scheduling, covering and packing problems, traveling salesman, knapsack, Steiner tree, resource constrained project scheduling and other discrete optimisation problems.

Bounds for the Quadratic Assignment Problem Using Continuous Optimization Techniques

Scott Hadley[1], Franz Rendl[2], and Henry Wolkowicz[1]

[1] Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada.
[2] Technische Universität Graz, Institut für Mathematik, Graz, Austria.

Abstract

The *quadratic assignment problem* (denoted QAP), in the trace formulation over the permutation matrices, is

$$\min_{X \in \Pi} \; tr(AXB + C)X^t.$$

Several recent lower bounds for QAP are discussed. These bounds are obtained by applying continuous optimisation techniques to approximations of this combinatorial optimisation problem, as well as by exploiting the special matrix structure of the problem. In particular, we apply constrained eigenvalue techniques, reduced gradient methods, subdifferential calculus, generalisations of trust region methods, and sequential quadratic programming.

Keywords : Quadratic Assignment Problem, Bounds, Constrained Eigenvalues, Reduced Gradient, Trust Regions, Sequential Quadratic Programming.

1 Introduction

The *quadratic assignment problem*, denoted QAP, is a generalization of the linear sum assignment problem, i.e. given the set $N = \{1, 2, \ldots, n\}$ and three n by n matrices $A = (a_{ik})$, $B = (B_{jl})$, and $C = (c_{ij})$, find a permutation π of the set N which minimizes

$$\sum_{i=1}^{n} c_{i\pi(i)} + \sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik} b_{\pi(k)\pi(i)}. \tag{1}$$

Equivalently, see [5], the trace formulation of QAP is to find a permutation matrix X to solve

$$\textbf{QAP} \qquad \min_{X \in \Pi} \; tr(AXB + C)X^t,$$

where Π denotes the set of $n \times n$ permutation matrices. The linear sum assignment problem, $LSAP$, can be solved very efficiently since the constraint can be relaxed from $X \in \Pi$ to $X \in \Omega$, the set of doubly stochastic matrices. This illustrates that $LSAP$ is in fact a linear programming problem. However, QAP is an NP-hard problem. (It contains, as a special case, the Travelling Salesman Problem, TSP.) If we relax the constraint as we did for $LSAP$, then there does not necessarily exist an optimum at an extreme point (i.e. permutation matrix) unless the objective function is concave. In the concave case, we may have many extreme points as local optima resulting in a hard combinatorial problem. Moreover, even the problem of finding an ϵ-approximation of the optimal solution is NP-hard. Thus from a worst case point of view, QAP is extremely difficult to solve. In the general case QAP is also difficult to solve, e.g. general QAP's of size $n \geq 15$ can prove to be intractable. Current solution techniques employ branch and bound methods and it appears that the poor quality of the lower bounds is a major cause of the difficulties. Thus it is worthwhile to collect a 'toolbox' of various bounds.

In this paper we present several bounds for QAP based on techniques from continuous optimization. These bounds exploit the special structure of QAP and include several interesting theoretical results. In Section 2 we present some of the background needed, as well as give a short survey of known bounds and solution techniques. In Section 3 we look at several bounding techniques based on constrained eigenvalue problems. In particular, we provide a projection technique onto the space of matrices with row and column sums 1. This is a special application of the reduced gradient technique in nonlinear programming. Our projection preserves orthogonality as well as the special matrix structure of our problem. This section also includes heuristic and iterative techniques for improving bounds. The iterative technique employs subgradient maximisation.

In Section 4 we show how to treat the nonsymmetric QAP, i.e. we provide a transformation from a nonsymmetric QAP to an Hermitian one, which allows the application of our eigenvalue bounds. This includes the introduction of an interesting class of matrices.

In Section 5 we apply generalisations of trust region techniques to QAP. This allows us to apply bounding techniques without splitting the objective function into two parts. Included is an interesting characterisation of global optimality for the QAP when the orthogonality constraint is relaxed using the Löewner partial order.

The results in this paper are taken from the following papers which have been recently submitted and/or are in progress: [8, 9, 10, 18, 19].

2 Background

In this section we present some background material including a short summary of results on QAP. We first present some of the notation and well known basic results used in this paper.

2.1 Notation and Basic Results

All matrices are assumed to be real and $n \times n$ unless stated otherwise. We let $u = \frac{1}{\sqrt{n}}(1, \ldots, 1)^t \in \Re^n$ be the normalized n-vector of ones, and u_i, $i = 1, \ldots, n$ be the unit vectors in $\Re^n$. We let $\mathcal{E}$ be the set of $n \times n$ matrices having row and column sums of one, $\mathcal{N}$ be the set of $n \times n$ matrices with non-negative elements, and $\mathcal{O}$ be the set of $n \times n$ orthogonal matrices. I.e.

$$\begin{aligned} \mathcal{E} &= \{X \in \Re^{n \times n} : Xu = X^t u = u\}; \\ \mathcal{N} &= \{X \in \Re^{n \times n} : X \geq 0 \ \textit{elementwise}\}; \\ \mathcal{O} &= \{X \in \Re^{n \times n} : XX^t = I\}. \end{aligned} \tag{2}$$

By definition, the set of *doubly stochastic* matrices, denoted Ω, satisfies

$$\Omega = \mathcal{E} \cap \mathcal{N}.$$

It is well known that the set of *permutation* matrices, denoted Π, satisfies

$$\Pi = \mathcal{O} \cap \Omega = \mathcal{O} \cap \mathcal{E} \cap \mathcal{N}. \tag{3}$$

Furthermore, $\Pi = \mathcal{O} \cap \mathcal{N}$. The *linear sum assignment problem* with cost coefficients $C = (c_{ij})$, denoted $LSAP(C)$, is

$$\textbf{LSAP(C)} \quad \min_{X \in \Pi} trCX^t.$$

LSAP can be solved as a linear program by relaxing the constraints $X \in \Pi$ to $X \in \Omega$.

Given matrix A, the sum of the rows, columns, and elements of A are denoted by

$$r_A = \sqrt{n}Au,\ c_A = \sqrt{n}A^t u,\ \textit{and}\ s_A = nu^t Au,$$

respectively.

We assume the matrices A and B are symmetric with orthogonal diagonalisations $\Lambda_1 = Q_1^t A Q_1$ and $\Lambda_2 = Q_2^t B Q_2$, respectively, and eigenvalues $\lambda = (\lambda_i)$ and $\mu = (\mu_i)$, respectively. (The results in this paper can be extended to the case where neither A nor B are symmetric, see e.g. [9, 10]. We discuss this in Section 4.) If the vectors λ and μ are arranged in non-decreasing order then we get the *maximal scalar product*

$$< \lambda, \mu >_+ = \sum_i \lambda_i \mu_i,$$

and the *minimal scalar product*

$$< \lambda, \mu >_- = \sum_i \lambda_i \mu_{n-i+1}.$$

We also use the Löewner partial order, i.e. for symmetric matrices $A \preceq B$ ($A \prec B$) denotes $B - A$ is positive semidefinite (resp. positive definite).

2.2 Solution Techniques for *QAP*

As mentioned in the introduction *QAP* is *NP-hard*. Unlike the *Travelling Salesman Problem (TSP)*, a special case of *QAP*, it is generally accepted that *QAP*'s of size $n > 15$ are intractable. This motivates the research of finding good, inexpensive, lower bounds for *QAP* to be implemented in a branch and bound procedure.

2.2.1 Lower Bounds for *QAP*

The classical lower bound for *QAP* is the Gilmore-Lawler bound [7, 13] which is based on a particular linear underestimation, $\tilde{C}$, of the objective function. The bound is given by the optimal function value of $LSAP(\tilde{C})$ and provides good bounds (relative to other known bounds). Unlike many other known bounds, the Gilmore-Lawler bound is not obtained by separating *QAP* into subproblems. We denote this bound by GLB.

In 1987, Finke, Burkard, and Rendl [5] introduced a new bound for *QAP* that is based on eigenvalue decompositions. Their bound is obtained by separating *QAP* into quadratic and linear subproblems. The linear subproblem can be solved exactly as an *LSAP*. The quadratic subproblem is relaxed to $X \in \mathcal{O}$ (from $X \in \Pi$). An optimal solution to the relaxed subproblem is characterised by the minimal scalar product of the eigenvalues of A and B. The bound is obtained by summing the optimal values of the relaxed subproblems, i.e.,

$$< \lambda, \mu >_- + LSAP(C). \tag{4}$$

We denote this eigenvalue based bound by *EVB1*.

The eigenvalue bound *EVB1* is, in general, very poor unless some reductions are applied to the matrices A and B. In order to maintain the symmetry of A and B, and the trace structure of the objective function of *QAP*, reductions are restricted to the following scheme (see e.g. [5]): Let $e, g, r, s \in \Re^n$ and define

$$E = \sqrt{n}(eu^t + ue^t);$$

$$G = \sqrt{n}(gu^t + ug^t);$$

$$R = diag(r);\ \textit{and}\ S = diag(s).$$

The two reduction rules we use are (see e.g. [5]):

[Rd1] Set $A = \tilde{A} + E$ and $B = \tilde{B} + G$. Then for every $X \in \Pi$,
$tr(AXB + C)X^t = tr(\tilde{A}X\tilde{B} + \tilde{C})X^t$ with
$\tilde{C} = C + 2(\tilde{A}G + EB)$.

[Rd2] Set $A = \tilde{A} + R$ and $B = \tilde{B} + S$. Then for every $X \in \Pi$,
$tr(AXB + C)X^t = tr(\tilde{A}X\tilde{B} + \tilde{C})X^t$ holds with
$\tilde{C} = C + (diag\tilde{A}) \cdot s^t + r \cdot (diag\tilde{B})^t + r \cdot s^t$.

Note that these reductions do not affect the objective function value of *QAP*, but in general affect the bounds. A reduction scheme that minimises the variance of the eigenvalues of A and B can be easily characterized, see e.g. [5, 18]. This scheme attempts to reduce the influence of the quadratic part of *QAP*, the part that is not solved exactly in *EVB1*. We denote the eigenvalue bound, applied after the variance minimizing reduction, by *EVB2*.

2.2.2 Exact Algorithms

To date, the most successful algorithms for *QAP* are branch and bound algorithms which, at any stage, consider a single assignment and either fix it or disallow it (i.e., set x_{ij} to either 1 or 0.). On a sequential machine, the most successful implementation is given in [1]; while the good implementations on parallel processors can be found in [17, 21].

There also exist algorithms based on cutting planes. Unlike *TSP* where cutting plane methods are very successful, these methods can only solve *QAP*'s of size less than or equal to 8. However, cutting plane methods have proved to be the basis for some of the best heuristics for *QAP*.

For a further discussion and survey of solution techniques see [10].

3 New Eigenvalue Based Bounds

In this section we discuss new lower bounds for *QAP* based on constrained eigenvalue techniques, see e.g. [8, 18]. These bounds improve on the eigenvalue bounds *EVB1* and *EVB2*.

3.1 Gradient Projection

A standard way of treating equality constraints in optimisation is known as *gradient projection* (or *reduced gradient*) methods. Consider the linearly constrained optimization problem

$$\begin{aligned} &min && f(x) \\ &subject\ to && Dx = b, \end{aligned} \tag{5}$$

and let Z be a matrix whose columns are a basis for the null space of D. Then (5) is equivalent to the unconstrained optimisation problem

$$\min_d \ f(x_0 + Zd)$$

for any x_0 such that $Dx_0 = b$.

We can similarly eliminate the linear equality constraints $X \in \mathcal{E}$ of *QAP*. This is done by exploiting the fact that u, the normalized n-vector of ones, is both a right and left eigenvector (with eigenvalue 1) of every permutation matrix X. Rather than relaxing to $X \in \mathcal{O}$ from $X \in \Pi$ as was done for *EVB1* and *EVB2*, we maintain the constraint $X \in \mathcal{E}$ by restricting ourselves to the orthogonal matrices acting on the orthogonal complement of u, $\{u\}^\perp$. Projecting *QAP* orthogonally onto $\{u\}^\perp$ we are able to obtain an

equivalent problem, denoted EP, see below. The objective function of EP has the same trace structure as QAP and also constrains the (matrix) variables to be orthogonal.

We need to define the following:

$$P := [u \vdots V], \text{ with } V^t u = 0 \text{ and } V^t V = I_{n-1}; \tag{6}$$

$Q := VV^t$ is the orthogonal projection on $\{u\}^\perp$;

$$P^tAP = \begin{bmatrix} \alpha & a^t \\ a & \hat{A} \end{bmatrix}; \; P^tBP = \begin{bmatrix} \beta & b^t \\ b & \hat{B} \end{bmatrix}; \; P^tCP = \begin{bmatrix} \gamma & c_1^t \\ c_2 & \hat{C} \end{bmatrix}; \tag{7}$$

where

$$\hat{A} = V^tAV; \; \hat{B} = V^tBV; \; \hat{C} = V^tCV;$$
$$a = V^tr_A/\sqrt{n}; \; b = V^tr_B/\sqrt{n};$$
$$c_1 = V^tr_{C^t}/\sqrt{n}; \; c_2 = V^tr_C/\sqrt{n};$$
$$\alpha = s_A/n; \; \beta = s_B/n; \; \gamma = s_C/n.$$

Now define the following programming problem in the matrix variable $Y \in \Re^{(n-1)\times(n-1)}$

$$\textbf{EP} \qquad \begin{array}{ll} min & tr(\hat{A}Y\hat{B}Y^t + (2ab^t + \hat{C})Y^t) + \alpha\beta + \gamma \\ subject\ to & YY^t = I, \; VYV^t \geq -uu^t. \end{array}$$

We will see that EP is equivalent to QAP. This is based on the following characterization of permutation matrices.

Lemma 3.1 *[8] $X \in \Pi$ if and only if*

$$X = PY_1P^t \text{ with } Y_1 = \begin{bmatrix} 1 & 0 \ldots 0 \\ 0 & \\ \vdots & Y \\ 0 & \end{bmatrix}, \tag{8}$$

for some orthogonal Y in $\Re^{(n-1)\times(n-1)}$ satisfying

$$VYV^t \geq -uu^t. \tag{9}$$

Using this lemma we are able to prove the following theorem.

Theorem 3.1 *[8] The matrix Y solves EP if and only if the matrix X defined by (8) solves QAP.*

Since EP and QAP are equivalent, a lower bound for QAP can be obtained by finding a lower bound for EP. As was done in [5] for $EVB1$ and $EVB2$, we separate EP into quadratic and linear subproblems, solve the linear subproblem exactly and solve the quadratic subproblem with the linear inequality constraints ignored. Relaxing the inequality constraints in EP is equivalent to considering orthogonal matrices having row and column sums of one in QAP. It is important to notice that the variables Y in EP are $(n-1)\times(n-1)$ matrices as oppposed to $n \times n$ matrices, i.e., the projection has reduced the dimension of the problem.

Theorem 3.2 *[8] Let $\hat{\lambda}$ and $\hat{\mu}$ be the eigenvalues of $\hat{A}$ and $\hat{B}$, repectively. Set*

$$D = V(2ab^t + \hat{C})V^t,$$

and define the LSAP

$$z^{opt} = \min_{X \in \Pi} trDX^t.$$

Then a lower bound for QAP is

$$< \hat{\lambda}, \hat{\mu} >_- + z^{opt} + (\alpha\beta + \gamma).$$

We denote the above lower bound by *IVB*. In the case that the linear term, C, of *QAP* is zero, the lower bound can be simplified, see e.g. [8],

$$< \hat{\lambda}, \hat{\mu} >_- + \frac{2}{n} < r_A, r_B >_- - \frac{s_A s_B}{n^2}. \tag{10}$$

The following theorem shows that reductions of type [Rd1], discussed in Section 2.2.1, do not affect the new bound. Therefore these shifts can be used to change characteristics of the data, e.g. the spectrum of A and B.

Theorem 3.3 *[8] Suppose that the reduction [Rd1] is applied to A and B. Then the lower bound in Theorem 3.2 is unchanged.*

We can however try to find a 'good' reduction using [Rd2] as was done for the bound in [5] and discussed in Section 2.2.1 above.

Theorem 3.4 *[8] The optimal real diagonal shift $A - R$, with $R = diag(r)$, that minimizes the variance of the eigenvalues of $V^t(A - R)V$ is defined by*

$$Q = VV^t;\ \delta = \left(\frac{n-1}{n}\right) trQA; \tag{11}$$

$$r_i = \frac{\delta}{n-1} + \frac{n}{n-1}(QAQ)_{ii} - \frac{1}{n}(trAQ).$$

3.2 Iterative Improvement

The eigenvalue bounds discussed above, *EVB1*, *EVB2*, *IVB*, rely on splitting *QAP* into linear and quadratic subproblems. Reductions may then be applied to attempt to increase the lower bound of the quadratic subproblem. The approach in [18] finds reductions to improve the lower bound by taking into account the effect of the reductions on *both* subproblems. Recall a lower bound for *QAP* can be obtained by the sum

$$m = < \tilde{\lambda}, \tilde{\mu} >_- + LSAP(\tilde{C}),$$

where $\tilde{A}$, $\tilde{B}$, and $\tilde{C}$ are given by [Rd1] and [Rd2], and $\tilde{\lambda}$, $\tilde{\mu}$ are the eigenvalues of $\tilde{A}$ and $\tilde{B}$, respectively. To improve the lower bound for *QAP*, (i.e. increase m), we use the derivative of the minimal scalar product, $< \lambda, \mu >_-$, as well as the subdifferential of the lower bound for $LSAP(C)$. A bundle and trust region subgradient routine from [22] was used to perform the maximization.

Assuming simple eigenvalues, the gradient of the minimal scalar product, say $d_1 = (e_1^t, g_1^t, r_1^t, s_1^t)^t$, can be obtained using the derivatives of the eigenvalues and the chain rule. (In the problems tested multiplicities of eigenvalues did not occur. The case of multiple eigenvalues can be handled using subdifferentials as well, see e.g. [16] for related problems.) The gradient of the lower bound for $LSAP(C)$ exists if the optimal solution

is unique. This is equivalent to saying the optimal basis does not change for sufficiently small perturbations of C. In the event the optimal basis changes, it is shown that the direction of steepest ascent (subgradient) for the linear subproblem, say d_2, is the element of minimal norm of the subdifferential, see e.g. [4, 20].

For problems of size $n \geq 12$, the bound obtained by this algorithm is the best to date. One drawback to this bound is that it is much more expensive to obtain than *GLB*, *EVB1*, *EVB2* and *IVB*. We denote this bound by *EVB3*.

3.3 Numerical Results

In this section we present a comparison of the eigenvalue based bounds, *EVB1*, *EVB2*, *EVB3*, *IVB*, with the Gilmore-Lawler bound *GLB*, on both random problems and standard test problems found in the literature.

The first 8 examples in table 1 are taken from [15] and the last 5 are taken from [2]. The examples from [15] were also used in [5, 18]. The examples from [2] are all of size $n = 10$ and were generated in the following way; the matrix A is symmetric with entries uniformly drawn from the integers 0,1,...,10, the matrix C also has entries drawn uniformly from the integers 0,1,...,10 but C is not necessarily symmetric. Finally the matrix B represents the squared Euclidean distances of 10 points drawn randomly from the integer lattice $(0, 1, ..., 10) \times (0, 1, ..., 10)$.

First note that the bound *IVB* outperforms the 'classical' bound *GLB* for problems of size $n \geq 15$. Comparing *EVB1* and *IVB* we see that the additional constraint $X \in \mathcal{E}$ drastically improves the lower bound. While the computation times of *GLB*, *EVB1*, *EVB2*, and *IVB* are comparable, each iteration of *EVB3* takes about the same amount of time. Thus we see that *IVB* is the best 'inexpensive' bound for *QAP*. Tests indicate that it takes approximately 20-30 iterations before *EVB3* outperforms *IVB*. However, each iteration of the iterative improvement bound *EVB3* is about as expensive as computing *IVB*. Moreover, it can be shown that there exists reductions of type [Rd1] and [Rd2] for *QAP* such that the new bound IVB is no worse than the bound EVB2, see [8].

	Best known		$X \in O$			$X \in O \cap \mathcal{E}$
Size n	Value	GLB	EVB1	EVB2	EVB3	IVB
5	50	50	-86	47	50	47
6	86	82	-160	70	70	69
7	148	137	-251	123	130	125
8	214	186	-409	160	174	167
12	578	493	-909	446	495	472
15	1150	963	-1745	927	989	973
20	2570	2057	-3198	2075	2229	2196
30	6124	4539	-7836	4982	5349	5265
10	4954	3586		2774	4541	4079
10	8082	6139		6365	7617	7211
10	8649	7030		6869	8233	7837
10	8843	6840		7314	8364	8006
10	9571	7627		8095	8987	8672

Table 1: Lower Bounds for *QAP*

4 Symmetrization and the Hermitian QAP

As mentioned above, the results in Section 3 can be extended to the case where both A and B are non-symmetric. In order to apply eigenvalue based bounds (see e.g. [5, 8, 18], Section 3), one must have real eigenvalues. Restricting ourselves to real matrices implies that A and B must be symmetric. However if we consider complex matrices we can relax the symmetric assumption to a Hermitian assumption. In [5] it is shown that if one of A or B is symmetric, then there exists an equivalent QAP with both matrices being symmetric. If we consider QAP as a quadratic form using the Kronecker product of A and B, then we can apply the standard symmetrisation using the transpose; however, we lose the important trace structure of our problem. In this section we show that given any (real) QAP there does exist an equivalent (complex) QAP such that the matrices A and B are Hermitian and C is real. The reductions discussed in Section 2.2.1 can be extended to the Hermitian case (and even to the quaternions), see [10]. This allows for extensions of eigenvalue based bounds to the *Hermitian-QAP* and hence permits the use of eigenvalue bounds for all real *QAPs*.

4.1 New Classes of Matrices

In order to describe how we transform a non-symmetric QAP to an equivalent *Hermitian-QAP* we need to describe several classes of matrices.

As is standard in the literature we let

$$A_+ = (A + A^*)/2,$$

denote the *Hermitian* part of A, and

$$A_- = (A - A^*)/2,$$

denote the *skew-Hermitian* part of A, where $\cdot^*$ denotes conjugate transpose. It immediately follows that $A = A_+ + A_-$. Letting $i = \sqrt{-1}$, we introduce two new classes of matrices. We define the *positive Hermitian* part of A as

$$\tilde{A}_+ = (A_+ + iA_-),$$

and the *negative Hermitian* part of A as

$$\tilde{A}_- = (A_+ - iA_-).$$

These new classes of matrices give rise to new results in matrix theory, e.g. an extension of the Hoffman-Wielandt inequality [11] to nonnormal matrices, see e.g. [9, 10].

Using these matrices and basic properties of the trace of a matrix we get the following theorem.

Theorem 4.1 *Given real matrices A, B, and C, the following problems are equivalent;*

$$QAP(A, B, C,),\ QAP(\tilde{A}_+, \tilde{B}_-, C),\ \textit{and}\ QAP(\tilde{A}_-, \tilde{B}_+, C).$$

This gives us the following lower bound for $QAP(A, B, C)$.

Theorem 4.2 *Given real matrices A, B, and C, a lower bound for $QAP(A, B, C)$ is*

$$< \tilde{\lambda}_+, \tilde{\mu}_- >_- + LSAP(C),$$

where $\tilde{\lambda}_+$ and $\tilde{\mu}_-$ are the eigenvalues of $\tilde{A}_+$ and $\tilde{B}_-$, respectively.

5 Further Nonlinear Programming Techniques

The bounds discussed in Section 3 dealt with the quadratic and linear parts of QAP separately. The iterative improvement technique of Section 3.2 attempted to bring these two parts back together. This outperformed all the other techniques as seen by our numerical tests. In this section we outline the approach taken in [19] which treats the two parts of QAP together. This involves the notions of trust regions, active sets, and sequential quadratic programming from nonlinear programming theory. Included is a characterization of optimality for a generalized trust region problem, see Theorem 5.1 below.

5.1 Trust Regions

The orthogonal constraint $XX^t = I$ resembles in form the quadratic vector norm constraint, $||x||^2 = x^t x = \delta_c^2$, $x \in \Re^n$, with $\delta_c = 1$. This latter constraint, when applied to the minimization of a quadratic form on $\Re^n$, results in an eigenvalue problem. Therefore, the eigenvalue bounds discussed above are intuitively not surprising. When δ_c is arbitrary, the norm constraint is often referred to as a trust region constraint, see e.g. [14]. Trust region algorithms are used for unconstrained minimization. Given the real, twice continuously differentiable, function $f(x)$ on $\Re^n$, a current estimate x_c, the gradient g_c at x_c, and an estimate of the Hessian H_c at x_c, then trust region approaches choose an approximate step length δ_c and find the next iterate x_+ as an approximate solution of the constrained quadratic model

$$\begin{aligned} \textit{min} \quad & f(x_c) + g_c^t d + \tfrac{1}{2} d^t H_c d \\ \textit{subject to} \quad & ||d|| \leq \delta_c. \end{aligned} \tag{12}$$

The radius of the trust region δ_c is increased or decreased depending on whether we get a satisfactory decrease in the objective function. The trust region subproblem (12) has several interesting properties. Surprisingly, it allows a characterisation of optimality independent of the convexity of the quadratic objective function, i.e. there is no gap between necessary and sufficient optimality conditions for a global minimum for (12). This holds for the equality constrained trust region problem $||d|| = \delta_c$ as well. Moreover, the characterization of optimality allows an approximation to the optimum to be obtained very efficiently. However, these nice properties do not follow through if we add additional trust region constraints. Even two trust regions prove to be a very hard problem, see e.g. [3].

Since the orthogonality constraint $XX^t = I$ resembles a trust region constraint, let us relax the orthogonality constraint to

$$XX^t \preceq I, \tag{13}$$

where $\preceq$ denotes the Löewner partial order. This constraint now resembles the vector norm constraint. However, the partial order constraint (13) requires many inequalities to be verified, i.e. it is equivalent to checking that all the principal minors of $XX^t - I$ are nonpositive. In [19] it is shown that, surprisingly, we get the following complete characterisation of optimality for the relaxation TR defined below. (We add a factor of $\frac{1}{2}$ to the quadratic part of QAP for notational convenience.)

Theorem 5.1 *Consider the relaxation of QAP*

$$\begin{aligned} \mathbf{TR} \qquad \textit{min} \quad & q(X) = tr \tfrac{1}{2} AXBX^t + CX^t \\ \textit{subject to} \quad & \tfrac{1}{2}(XX^t - I) \preceq 0. \end{aligned}$$

Suppose that $XX^t \preceq I$. Define the optimality conditions

$$(i) \qquad S \succeq 0,$$

$$(ii) \qquad AXB + C + SX = 0, \tag{14}$$

$$(iii) \qquad tr\ S(XX^t - I) = 0,$$

$$(iv) \quad tr\ (AhBh^t + hSh^t) \geq 0, \ \textit{if the matrix } h \textit{ satisfies } Xh^t + hX^t \textit{ is psd on } \mathcal{N}(XX^t - I).$$

Then:

a) X is a global minimum for TR *if and only if (14)(i)-(iv) holds;*

b) if (14)(i)-(iv) holds with $S \succ 0$, then X is a global minimum of TR *and $XX^t = I$.*

We can, in addition, guarantee that the optimum of *TR* is orthogonal, i.e. we can replace the constraint $XX^t \preceq I$ with $XX^t = I$. This involves applying [Rd2] reductions to guarantee the singular value condition in the following.

Corollary 5.1 *Suppose that the smallest singular value*

$$\sigma_n(A^{-1}CB^{-1}) > 1.$$

Then the second order conditions (14) characterize optimality of X for TR, with X orthogonal, i.e. X orthogonal solves TR if and only if the first order conditions (14)(i)-(iii) hold and

$$tr(AhBh^t + hSh^t) \geq 0 \ if \ Xh^t + hX^t \succeq 0.$$

The above characterisations of a global optimum of *TR* are of theoretical interest in that they extend the vector trust region results which hold only for a single trust region. It remains to develope fast approximate solution techniques based on these characterisations, just as was done for vector trust region algorithms for unconstrained minimisation, see e.g. [14].

5.2 Sequential Quadratic Programming and Active Sets

In Section 3.2 we introduced a projection onto $\mathcal{E}$, the smallest linear manifold containing the matrices with row and column sums 1. This yielded the following equivalent problem to QAP

$$\textbf{EP} \qquad \begin{array}{ll} min & tr(\hat{A}Y\hat{B}Y^t + (2ab^t + \hat{C})Y^t) + \alpha\beta + \gamma \\ subject\ to & YY^t = I,\ VYV^t \geq -uu^t, \end{array}$$

where the optimal permutation for QAP and the optimal orthogonal Y for EP are related by

$$X = P \begin{bmatrix} 1 & \\ & Y \end{bmatrix} P^t;\ P = [u \vdots V];\ PP^t = I.$$

The projection eliminated the constraints of $\mathcal{E}$ while maintaining the special trace structure as well as the orthogonality constraint of our problem. Being able to maintain this structure is what enabled us to obtain the eigenvalue bound *IVB*. The constraint $VYV^t \geq -uu^t$ corresponds to the nonnegativity constraint of QAP, i.e. $X \geq 0$. Thus

$$X_{ij} = 0 \ \textit{if and only if} \ V_iYV_j^t = \frac{-1}{n}, \tag{15}$$

where V_k is the *k-th* row of V.

To solve *EP* we need to apply the trust region results presented above with a branch and bound procedure applied to an active set approach. The active set is obtained by fixing one of the branches $X_{ij} = 1$ or $X_{ij} = 0$. If $X_{ij} = 1$ is fixed, then we can simultaneously fix the rest of the *i-th* row and *j-th* column of X to 0. This results in a *QAP* of smaller dimension.

Handling $X_{ij} = 0$ fixed is somewhat harder. We first apply the projection and work with the equivalent problem EP and the constraint $V_i Y V_j^t = \frac{-1}{n}$ fixed. Let $V_k^t = U_k \Sigma_k$ be the singular value decomposition of the vector V_k^t; thus we have $U_k^t U_k = I$ and $\Sigma_k = (\sigma_k\, 0 \ldots 0)^t \in \Re^{(n-1)}$. Substitute $Y = U_i Z U_j^t$ in EP to get a new equivalent program in $Z \in \mathcal{O}$ with the added constraint that $Z_{11} = \frac{-1}{\sigma_i \sigma_j}$ corresponding to $X_{ij} = 0$. This problem can be handled with a sequential quadratic programming technique, where the orthogonality constraint $ZZ^t = I$ can be handled as an eigenvalue constraint on $ZZ^t - I$, with the Z_{11} element of Z fixed. Thus, when we treat the inequality constraints in EP with an active set approach in this way, we can maintain both the orthogonality and trace structures. This allows us to apply eigenvalue based bounds. However, we can do even more by substituting the matrix exponential $Z = e^{iT}$, where $i = \sqrt{-1}$ and $T = T^t$ is real and symmetric. (The parametrization $Z = e^T$ has been used for the orthogonal diagonalization of matrices in [6].) This parametrization of the orthogonal matrices actually yields all the complex orthogonal matrices, see [12] Problem 27.6.4. This eliminates all the constraints except for the nonnegativity ones. If we relax our problem by ignoring the nonnegativity constraints only, then we are left with an unconstrained minimization problem in only $n(n+1)/2$ variables and we have not split up the objective function into linear and quadratic parts. We can also apply reductions to improve this bound, i.e. we are left with the max-min (or parametric minimization over the reduction vectors e, g, r, s) problem

$$\max_{e,g,r,s} \min_{T=T^t} \; Re\{tr(\hat{A}w(T)\hat{B}w(T)^* + (2ab^t + \hat{C})w(T))\} + \alpha\beta + \gamma, \tag{16}$$

where $w(T) = e^{iT}$, and all data are functions of the reductions.

References

[1] R. E. Burkard and U. Derigs. *Assignment and Matching Problems: Solution Methods with Fortran Programs.* Springer-Berlin, 1980.

[2] P. Carraresi and F. Malucelli. A new lower bound for the quadratic assignment problem. Technical Report TR-7/88, Universita di Pisa, 1988.

[3] M.R. Celis, J.E. Dennis Jr., and R.A. Tapia. A convenient trust region subproblem for nonlinear programming. *Presented at ORSA/TIMS meeting, New York*, 1989.

[4] F. H. Clarke. *Optimization and Nonsmooth Analysis.* Wiley-Interscience, 1983.

[5] G. Finke, R. E. Burkard, and F. Rendl. Quadratic assignment problems. *Annals of Discrete Mathematics*, 31:61–82, 1987.

[6] S. Friedland, J. Nocedal, and M.L. Overton. The formulation and analysis of numerical methods for inverse eigenvalue problems. *SIAM J. Numer. Anal.*, 24:634–667, 1987.

[7] P.C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics*, 10:305–313, 1962.

[8] S. W. Hadley, F. Rendl, and H. Wolkowicz. A new lower bound via projection for the quadratic assignment problem. Technical Report CORR 89-5, University of Waterloo, Waterloo, Ontario, 1989.

[9] S. W. Hadley, F. Rendl, and H. Wolkowicz. Symmetrization of nonsymmetric quadratic assignment problems and the Hoffman-Wielandt inequality. Technical Report CORR 89-5, University of Waterloo, Waterloo, Ontario, 1989.

[10] S.W. Hadley. *Continuous Optimization Approaches for the Quadratic Assignment Problem.* PhD thesis, University of Waterloo, 1989.

[11] A.J. Hoffman and H.W. Wielandt. The variation of the spectrum of a normal matrix. *Duke Mathematics*, 20:37–39, 1953.

[12] R. Horn and C. Johnson. *Matrix Analysis.* Cambridge University Press, New York, 1985.

[13] E. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.

[14] J.J. More and D.C. Sorensen. Computing a trust region step. *SIAM J. Sci. Stat. Comput.*, 4:553–572, 1983.

[15] C.E. Nugent, T.E. Vollman, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16:150–173, 1968.

[16] M.L. Overton. On minimizing the maximum eigenvalue of a symmetric matrix. *SIAM J. Matrix Anal. Applic.*, 9:256–268, 1988.

[17] P.M. Pardalos and J.V. Crouse. A parallel algorithm for the quadratic assignment problem. Technical Report, The Pennsylvania State University, University Park, PA., 1989.

[18] F. Rendl and H. Wolkowicz. Applications of parametric programming and eigenvalue maximization to the quadratic assignment problem. *Mathematical Programming*, 1989. To appear.

[19] F. Rendl and H. Wolkowicz. Trust region methods for the quadratic assignment problem. Technical Report, University of Waterloo, 1990. In progress.

[20] R.T. Rockafellar. *Convex Analysis.* Princeton University Press, 1970.

[21] C. Roucairol. A parallel branch and bound algorithm for the quadratic assignment problem. *Discrete Applied Mathematics*, 15:211–225, 1987.

[22] H. Schramm and J. Zowe. A combination of the bundle approach and the trust region concept. Technical Report 20, SPP der DFG - Anwendungsbezogene Optimierung und Steuerung, 1987.

Near-Optimal Sequencing with Precedence Constraints

Leslie A. Hall
Princeton University

David B. Shmoys *
Cornell University

Abstract

We consider the following scheduling problem: each of n jobs is to be scheduled without interruption on a single machine. Each job has a release date, when it first becomes available for processing, and, after completing its processing on the machine, requires an additional delivery time. Feasible schedules are further restricted by job precedence constraints, and the objective is to minimize the time by which all jobs are delivered. In the notation of Graham, Lawler, Lenstra and Rinnooy Kan, this problem is denoted $1|r_j, prec|L_{max}$. We show that, for each positive relative error ϵ, we can construct a polynomial-time algorithm that is guaranteed to deliver a feasible schedule no longer than $(1+\epsilon)$ times the length of the optimal schedule.

1 Introduction

Precedence constraints arise in a number of practical scheduling problems, and one important area of research in scheduling theory has been to study under what conditions a precedence-constrained problem is computationally harder than its counterpart with independent jobs. We shall consider this question with respect to finding provably near-optimal solutions for strongly $\mathcal{NP}$-hard problems; for a particular problem, what is the best relative error that can be guaranteed by a polynomial-time algorithm, or can *any* performance guarantee be achieved? Typically, precedence constraints make a problem provably harder. For example, scheduling unit-length jobs on identical parallel machines to minimize the maximum completion time is a trivial problem; just balance the number of jobs assigned to each machine as much as possible. On the other hand, for the same problem except with a precedence relation constraining feasible schedules, Lenstra & Rinnooy Kan [6] have shown that no polynomial-time algorithm can guarantee a relative error better than 1/3, unless $\mathcal{P} = \mathcal{NP}$. In this paper, we give the first example of a precedence-constrained scheduling problem that is (strongly) $\mathcal{NP}$-hard, and yet any fixed positive relative error can be guaranteed in polynomial time.

We shall study the following problem. Each job j, $j = 1, \ldots, n$, must be processed without interruption for p_j time units on a single machine, which can process at most one job at a time. Job j must begin processing no earlier than its *release date* r_j, and the schedule must also satisfy precedence constraints given by the partial order $\prec$, where $j \prec k$ means that job k must be processed after job j. Following processing, job j requires delivery time q_j; if σ_j denotes the time j starts processing, it is has been delivered at time $D_j = \sigma_j + p_j + q_j$. Our objective is to minimize, over all possible schedules, $D_{max} = \max_j D_j$; this minimum shall be denoted D^*_{max}. Delivery is a *non-bottleneck* activity, in that all jobs may be simultaneously delivered. Alternatively, a delivery time q_j for job j may be viewed as its due date $-q_j$, and then D_j is the difference between the time that

*Research partially supported by NSF PYI Grant CCR-89-96272 with matching funds from IBM, UPS, and Sun and by the Cornell Computational Optimization Project.

j completes processing and its due date, or in other words, its *lateness*, L_j. Thus, in the notation of Graham, Lawler, Lenstra & Rinnooy Kan [1], this problem is denoted $1|r_j, prec|L_{max}$. We shall assume that all data are integral.

In an earlier paper [2], we showed that there is a simple 4/3-approximation algorithm for $1|r_j, prec|L_{max}$, *i.e.*, a polynomial-time algorithm that always delivers a schedule Σ with $D_{max}(\Sigma) \leq (4/3)D^*_{max}$. However, without precedence constraints, we were able to give a *polynomial approximation scheme*, a family of algorithms $\{A_\epsilon\}$ such that, for any $\epsilon > 0$, A_ϵ is a $(1+\epsilon)$-approximation algorithm. The main result of this paper is to present a polynomial approximation scheme for $1|r_j, prec|L_{max}$, by extending the techniques used for the case of independent jobs.

The new result highlights an interesting distinction between $1|r_j|L_{max}$ and its generalization that allows each job to be processed on one of m parallel identical machines. In another paper [3], we have generalized the scheme for $1|r_j|L_{max}$ to this setting, as well; however, if precedence constraints are added to this parallel machine problem, one can prove that no ρ-approximation algorithm exists for any $\rho < 4/3$ (unless $\mathcal{P} = \mathcal{NP}$) [6]. Thus, the new result provides evidence that the parallel version is harder than the single-machine problem.

For polynomially-solvable scheduling problems, one of the most effective techniques for handling precedence constraints is a preprocessing step generally known as *due-date modification*. As an example, consider the special case of $1|prec, r_j|L_{max}$ in which all $p_j = 1$. Without precedence constraints, this can be solved by the following on-line algorithm to construct a schedule: among all jobs already released, schedule next the job with the longest delivery time. (Since this generalizes a procedure to solve $1||L_{max}$ due to Jackson [4], it is often called the extended Jackson's rule; this procedure was also suggested by Schrage as a heuristic for $1|r_j|L_{max}$.)

Lageweg, Lenstra & Rinnooy Kan [5] observed that essentially the same algorithm can be used to solve $1|r_j, prec, p_j = 1|L_{max}$, after a suitable preprocessing of the data. Consider the effect of applying the extended Jackson's rule to an instance that satisfies the property that

$$j \prec k \Rightarrow (r_j \leq r_k \text{ and } q_j \geq q_k). \tag{1}$$

(One additional clarification is needed: if more than one job can be selected according to this rule, schedule a job whose predecessors have all been scheduled already.) A simple proof by contradiction will show that the resulting schedule must satisfy the precedence constraints. Suppose that $j \prec k$ and that k is scheduled before j. Since $q_j \geq q_k$, then j must not have been released at the time that k is scheduled; thus $r_k < r_j$ and we have reached a contradiction.

However, if $j \prec k$ and $r_j > r_k$, then we can reset $r_k := r_j$ and each feasible schedule will remain feasible. Similarly, if $q_j < q_k$ then we can reset $q_j := q_k$ without changing the objective function value of any feasible schedule. Thus, by repeatedly applying these updates, we can always obtain an equivalent instance that satisfies (1). If the extended Jackson's rule is applied to this modified instance, we get the optimal solution, which automatically satisfies the precedence constraints.

There are more efficient ways to implement this preprocessing of the data than described above. As before, update the release dates and delivery times separately. Let $j_1, j_2, \ldots, j_n$ denote any fixed ordering of the jobs that is consistent with the precedence relation; this can be obtained by topologically sorting the precedence graph. To update the release dates, the jobs are processed in this order; when job k is processed, then r_k is reset to the maximum of its original value and r_j for any job j with $j \prec k$ (where r_j refers to the updated release date for j). To update the delivery times, the jobs are processed in the reverse order; when job j is processed, then q_j is reset to the maximum of its original value and q_k for any job k such that $j \prec k$ (where q_k refers to the updated delivery time for k). These changes ensure that the resulting times satisfy (1). We shall refer to

this as the LLR procedure. (In fact, the original procedure of Lageweg, Lenstra & Rinnooy Kan yielded an equivalent instance with somewhat stronger properties, but this shall be sufficient for our purposes.)

We will show that preprocessing techniques are also useful in the context of approximation algorithms. The 4/3-approximation algorithm for $1|prec, r_j|L_{max}$ mentioned above follows this approach, as does the scheme presented below. The 4/3-approximation algorithm for $1|r_j|L_{max}$ works by running the extended Jackson's rule on a series of instances and choosing the best solution. It is not too hard to see that by preprocessing each of these instances, we can extend this algorithm to handle precedence constraints. The difficulty in extending this plan of attack to obtain a polynomial approximation scheme for $1|prec, r_j|L_{max}$ is that all known schemes for this problem use techniques more sophisticated than the extended Jackson's rule, and so the preprocessing and the heuristic no longer combine to enforce the precedence constraints. We show that a more sophisticated preprocessing algorithm can be used to extend a scheme for $1|r_j|L_{max}$ to handle precedence constraints.

2 Without Precedence Constraints: A Quick Review

The polynomial approximation scheme for $1|r_j, prec|L_{max}$ is based on a technique introduced for $1|r_j|L_{max}$, called an *outline scheme*.

Definition. An *outline scheme* for a problem Π is a labeling of feasible solutions such that, for each feasible solution x, the associated label ω, called an *outline*, provides concise information about x; more specifically, the length of ω is bounded by a polynomial function of the length of x.

For example, one outline scheme for a scheduling problem labels each job with its starting time in the schedule. Of course, this outline is particularly useful because it completely describes the original schedule. Most outlines reveal only partial information about their corresponding schedule or schedules; even so, this partial information might still be enough to reconstruct a schedule that is "close to", or almost as good as any schedule corresponding to the outline.

Definition. For $\epsilon > 0$, a $(1+\epsilon)$-*approximate outline scheme* for Π is an outline scheme for Π and a polynomial-time algorithm $\mathcal{A}$ with the following property: given any outline ω for an instance $\mathcal{I}$, algorithm $\mathcal{A}$ delivers a feasible solution to $\mathcal{I}$ of value at most $(1+\epsilon)$ times the value of any feasible solution of $\mathcal{I}$ that is labeled with ω.

In particular, given an outline corresponding to the optimal solution to $\mathcal{I}$, the algorithm delivers a solution of value at most $(1+\epsilon)$ times the optimum. Moreover, if for every instance $\mathcal{I}$ of Π, the number of distinct outlines associated with a $(1+\epsilon)$-approximate outline scheme is polynomial in the size of $\mathcal{I}$, then by running the algorithm on every outline for $\mathcal{I}$, we obtain a $(1+\epsilon)$ -approximation algorithm for Π.

The first step in obtaining the approximation scheme for $1|r_j|L_{max}$ is to show that, without loss of generality, we can restrict attention to instances with a fixed number κ of distinct release-date values.

Lemma 1 If, for any positive integer κ, there exists a polynomial approximation scheme for the special case of $1|r_j|L_{max}$ in which the number of distinct release dates is at most κ, then there is a polynomial approximation scheme for $1|r_j|L_{max}$.

Proof: We need to show that for any fixed $\epsilon > 0$, there is a polynomial-time $(1+\epsilon)$-approximation algorithm for the unrestricted version. Let $\Phi = \max\{\sum_j p_j, \max_j\{r_j\}\}$ and consider the following algorithm. Round each release date down to the nearest multiple of $\Phi\epsilon/2$. Note that this modification does not increase the optimal objective-function value. The modified input has at most $1 + 2/\epsilon$ distinct release dates, and so we can apply a $(1+\epsilon/2)$-approximation algorithm that can handle $1+2/\epsilon$ distinct release dates. This yields a schedule of length at most $(1+\epsilon/2)D^*_{max}$, where D^*_{max} denotes the optimal schedule length for the original instance. Adding $\Phi\epsilon/2$ to the each job's starting time in this schedule yields a feasible schedule for the original instance, and increases D_{max} by at most $\Phi\epsilon/2 \leq (\epsilon/2)D^*_{max}$. Therefore, the two steps together yield a $(1+\epsilon)$-approximation algorithm. ∎

Observe that this reduction applies equally to $1|prec, r_j|L_{max}$. We will assume henceforth that there is a fixed number κ of release dates, which we denote $\rho_1 < \rho_2 < \cdots < \rho_\kappa$, and $\rho_{\kappa+1}$ shall denote $+\infty$. Note that without loss of generality, we can still assume that all data are integral.

Next, for any $\epsilon > 0$, we show how to construct a $(1+\epsilon)$-approximate outline scheme. Let $\delta = \Phi\epsilon/2\kappa$. We divide the jobs into two sets, according to the length of their processing times, $A = \{j : p_j \geq \delta\}$ and $B = \{j : p_j < \delta\}$. Observe that for any fixed ϵ, there is only a constant number of jobs in A. Consider a particular feasible schedule Σ given by starting times $\sigma_1, \ldots, \sigma_n$. If $\rho_i \leq \sigma_j < \rho_{i+1}$, then we call ρ_i the *effective release date* of job j and denote it by $\overline{r}_j$. (More properly, we should denote this by $\overline{r}_j(\Sigma)$ to reflect the dependence on the choice of Σ; however, for this and the notation below, we believe that the appropriate Σ will be clear from the context, and thus we omit it.) Define $I_i = \{j \in A : \overline{r}_j = \rho_i\}$ and $H_i = \{j \in B : \overline{r}_j = \rho_i\}$. Let $M_i = \sum(p_j : j \in H_i)$, and set $N_i = \lceil M_i/\delta \rceil \delta$, for $i = 1, \ldots, \kappa$; that is, N_i is the approximate amount of time in the interval $[\rho_i, \rho_{i+1})$ that is spent processing the small jobs.

We will show that the labeling $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$ is sufficient to complete the $(1+\epsilon)$-approximate outline scheme. The algorithm *Expand* converts such an outline into a good schedule as follows (see also Figure 1). For each interval $[\rho_i, \rho_{i+1})$, we construct a set of jobs that, ideally, we would like to start processing within that interval. Each of these sets is ordered by using Jackson's rule (*i.e.*, schedule the jobs in order of non-increasing delivery time), and then the entire schedule is formed by concatenating each of these pieces together. We construct the sets iteratively, starting with the earliest interval first. Suppose that the first $i-1$ sets have been constructed. Clearly, I_i should be scheduled in the ith interval. In addition, consider the set of short jobs that have not yet been assigned to a set, but have been released by time ρ_i. Sort these jobs by nondecreasing q_j, and choose the minimal prefix of this sequence with total processing time at least N_i; call this set J_i. (If the total processing time of these jobs is less than N_i, assign all of them to this interval.)

It is useful to realize that the schedule generated by a particular outline need not (and probably will not) have the same outline itself. In order to analyze the performance of this scheme, we must compare D_{max} for the schedule output by *Expand* on input $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$ to D_{max} for a schedule Σ that is labeled with outline $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$. Without loss of generality, we can focus on a certain *canonical* schedule that is associated with that outline. First, we may assume that Σ is an optimal schedule subject to the constraint that it has outline $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$. (If the schedule produced by *Expand* is good compared to such a schedule, then it is good compared to any schedule consistent with the given outline.) Furthermore, we can assume that the schedule is minimal in the sense that if the processor is idle immediately preceding the starting time σ_j of job j, then job j is released at σ_j, and so cannot start processing earlier. (Note that starting j earlier need not decrease D_{max}.) If ω is an outline, then $D^*_{max}(\omega)$ denotes the length of this optimal schedule consistent with ω.

Algorithm Expand$(\{(I_i, N_i) : i = 1, \ldots, \kappa\})$

$J_i := \emptyset$, $i = 1, \ldots, \kappa$;
for $i := 1, \ldots, \kappa$
  $W_i := \{j \in B \ : \ j \notin J_k$, for any $k < i$, and $r_j \leq \rho_i\}$;
  $m := |W_i|$;
  let $j_1, \ldots, j_m$ denote the jobs in W_i, such that $q_{j_1} \geq \cdots \geq q_{j_m}$;
  if $\sum_{j \in W_i} p_j < N_i$ **then let** $J_i := W_i$;
    else $J_i :=$ the minimal set $\{j_1, \ldots, j_l\}$ such that $\sum_{j \in J_i} p_j \geq N_i$;
  sort the jobs in $I_i \cup J_i$ in order of non-increasing q_j;
schedule all jobs as follows: first schedule the jobs in $I_1 \cup J_1$ in the order determined above,
  then schedule the jobs in $I_2 \cup J_2$, and so on, through $I_\kappa \cup J_\kappa$.

Figure 1: *Algorithm Expand*

Theorem 1 The labeling $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$, along with algorithm *Expand*, is a $(1 + \epsilon)$-approximate outline scheme for $1|r_j|L_{max}$. Furthermore, if Σ' denotes the schedule delivered by algorithm *Expand* when given the outline ω,

$$D_{max}(\Sigma') \leq D^*_{max}(\omega) + \epsilon\Phi. \tag{2}$$

The proof of this theorem is based upon the following two lemmas, which are central to our extension, as well. The first is a type of domination result, that says that the greedy strategy employed really does guarantee that at each interval, the problem remaining unscheduled is "no harder" than the problem remaining for any schedule that generated this outline. The proof, which we omit, follows from a simple inductive argument, and can be found in [2].

Lemma 2 For any $q \geq 0$, and any index i, $1 \leq i \leq \kappa$,

$$\sum_{l=1}^{i} \sum_{j \in H_l} (p_j : q_j \geq q) \leq \sum_{l=1}^{i} \sum_{j \in J_l} (p_j : q_j \geq q). \tag{3}$$

Let $\overline{p}_i$ denote $\min\{\sigma_j : \overline{r}_j \geq \rho_i\}$, which indicates the time when the schedule Σ begins processing jobs solely in the interval $[\rho_i, \infty)$. Consider the quantity

$$\overline{p}'_i = \max\{\rho_i, \max_{s=1,\ldots,i-1}\{\rho_s + \sum_{l=s}^{i-1}(N_l + \delta + \sum_{k \in I_l} p_k)\}\}.$$

This indicates the time that, according to the outline, the processor is guaranteed to be available to schedule jobs designated to be in the ith interval; the second lemma says that $\overline{p}'_i$ serves as a good estimate for $\overline{p}_i$.

Lemma 3 If $I_i \neq \emptyset$ or $N_i \neq 0$, then $\overline{p}_i \leq \overline{p}'_i < \overline{p}_i + 2\delta(\kappa - 1)$.

Proof: To prove the lemma, we first give an alternative characterization of $\overline{p}_i$. Note that $I_i \neq \emptyset$ or $N_i \neq 0$ implies that the schedule Σ with this label has a job that is scheduled to begin within the

interval $[\rho_i, \rho_{i+1})$. Let j be the job started earliest within this interval (and so $\sigma_j = \overline{\rho}_i$). Since Σ is a canonical schedule, if we look for the latest time that the processor is idle prior to σ_j, this idle time ends at some release time ρ_s. This implies that

$$\begin{aligned}\overline{\rho}_i &= \max\{\rho_i, \max_{s=1,\ldots,i-1}\{\rho_s + \sum_{l=s}^{i-1}\sum_{j\in I_l\cup H_l} p_j\}\} \\ &= \max\{\rho_i, \max_{s=1,\ldots,i-1}\{\rho_s + \sum_{l=s}^{i-1}(M_l + \sum_{j\in I_l} p_j)\}\}.\end{aligned}$$

Since $M_l \le N_l + \delta \le M_l + 2\delta$, it is easy to see that, in fact,

$$\overline{\rho}_i \le \overline{\rho}'_i \le \overline{\rho}_i + 2\delta(i-1),$$

which completes the proof. ∎

We are now ready to give the proof of Theorem 1.

Proof of Theorem 1: We will prove that in the schedule Σ' produced by *Expand*, each job j is delivered before time $D^*_{max}(\omega) + 2\kappa\delta = D^*_{max}(\omega) + \epsilon\Phi$. Focus on a particular job h, and suppose that $h \in I_i \cup J_i$; that is, h is scheduled to start processing in the ith interval in Σ'. If we set

$$\tau_A = \sum(p_j \ : \ j \in I_i \text{ and } q_j \ge q_h),$$

and

$$\tau_B = \sum(p_j \ : \ j \in J_i \text{ and } q_j \ge q_h),$$

then

$$D_h \le \overline{\rho}'_i + \tau_A + \tau_B + q_h. \tag{4}$$

We will show that in the schedule Σ, there is a set of jobs with total processing time at least $\tau_A + \tau_B$, each of which has delivery time at least q_h and starts processing no earlier than $\overline{\rho}_i$. By applying Lemma 5 with delivery time threshold q_h and index $i-1$, we see that

$$\sum_{l=1}^{i-1}\sum_{j\in H_l}(p_j : q_j \ge q_h) \le \sum_{l=1}^{i-1}\sum_{j\in J_l}(p_j : q_j \ge q_h).$$

This is equivalent to

$$\sum_{l=i}^{\kappa}\sum_{j\in H_l}(p_j : q_j \ge q_h) \ge \sum_{l=i}^{\kappa}\sum_{j\in J_l}(p_j : q_j \ge q_h).$$

Clearly,

$$\sum_{l=i}^{\kappa}\sum_{j\in J_l}(p_j : q_j \ge q_h) \ge \sum_{j\in J_i}(p_j : q_j \ge q_h) = \tau_B.$$

By combining these inequalities, we have shown that there is a subset of jobs in B that all begin processing in Σ after $\overline{\rho}_i$, have delivery time at least q_h, and have total processing time at least τ_B. In addition, since each job in I_i is scheduled in Σ to begin no earlier than $\overline{\rho}_i$ (by the definition of $\overline{\rho}_i$), we see that each job that contributes to τ_A begins processing no earlier than $\overline{\rho}_i$. Thus, the last one of all of these jobs to be processed in Σ completes its processing no earlier than $\overline{\rho}_i + \tau_A + \tau_B$, and therefore the length of Σ satisfies

$$D^*_{max}(\omega) \ge \overline{\rho}_i + \tau_A + \tau_B + q_h. \tag{5}$$

Combining (5) with (4), we have

$$D_h \leq \overline{p}_i + \tau_A + \tau_B + q_h + (\overline{p}'_i - \overline{p}_i) \leq D^*_{max}(\omega) + (\overline{p}'_i - \overline{p}_i),$$

and by applying Lemma 3, we get

$$D_h \leq D^*_{max}(\omega) + 2\delta(\kappa - 1). \quad \blacksquare$$

Given such a family of $(1+\epsilon)$-approximate outline schemes, to complete the polynomial approximation scheme it is only necessary to observe that, in fact, there is only a constant number of different outlines. Thus, in polynomial time, it is possible to run the appropriate outline scheme on each possible outline and choose the best schedule found. Since one of the outlines corresponds to the optimum, we obtain the desired result.

3 Handling Precedence Constraints

We have seen that, by preprocessing the data to obtain an equivalent problem, it is sometimes possible that a particular algorithm will generate a schedule that automatically satisfies the precedence constraints. Ideally, we would like to run such a preprocessing procedure on our instance, and then directly apply algorithm *Expand* given in the previous section. In fact, *Expand* makes critical use of the extended Jackson's rule, and so there is some reason to believe that this approach might be successful. Of course, the algorithm *Expand*, as given in Figure 1, will certainly require a clarification analogous to the one mentioned for the extended Jackson's rule: whenever jobs are sorted in order of non-increasing delivery times, ties are broken in a way consistent with the precedence constraints.

A simple way to formalize the resemblence of *Expand* to the extended Jackson's rule is summarized by the following lemma; it is an immediate consequence of the fact that algorithm *Expand* constructs its schedule by using the extended Jackson's rule for each set $I_i \cup J_i$ and concatenating these pieces.

Lemma 4 Consider an instance $\mathcal{I}$ satisfying property (1), and an outline ω. Suppose that $j \prec k$, and algorithm *Expand* has assigned these two jobs such that $j \in I_s \cup J_s$ and $k \in I_t \cup J_t$ where $s \leq t$. Then algorithm *Expand* schedules j before k.

Unfortunately, it is not hard to see that (1), which could be enforced by the LLR procedure, will *not* be sufficient to ensure that all precedence constraints are obeyed by the resulting schedule. For example, there is no reason that if $j \in B$, $k \in I_i$, and $j \prec k$, then the algorithm will assign j to a sufficiently early interval. However, we will show that relatively simple changes to the LLR procedure will suffice.

In essence, the cause of the problem is that the algorithm *Expand*, while based on the extended Jackson's rule, *uses* the outline in producing the schedule; the LLR procedure, which does not use it, cannot foresee the effect that the outline will have in constraining the schedules being considered. As this suggests, a natural idea is to consider a preprocessing procedure that also makes use of the outline. In fact, our modified preprocessing procedure will have two stages: one that primarily focuses on the way the outline affects the release dates, and another for the delivery times.

When the outline specifies that $j \in I_i$, it constrains the algorithm *Expand* to schedule j after ρ_i. Thus, it is natural to update r_j to ρ_i. This suggests the following procedure *Update-by-release-dates*, which is given an instance of $1|prec, r_j|L_{max}$ along with an outline $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$:

1. For all $j \in I_i$, set $r_j := \rho_i$, $i = 1, \ldots, \kappa$.

2. Run the LLR procedure.

Note that release dates do not become smaller in either step of this procedure. Although this procedure is not sufficient to ensure that algorithm *Expand* will automatically enforce all of the precedence constraints, the next two lemmas show that this procedure takes a big step towards achieving that goal.

Lemma 5 Consider an instance $\mathcal{I}$ and an outline $\omega = \{(I_i, N_i) : i = 1, \ldots, \kappa\}$ corresponding to some feasible schedule Σ for $\mathcal{I}$. Suppose that $\mathcal{I}$ satisfies property (1) and for $i = 1, \ldots, \kappa$,

$$j \in I_i \Rightarrow r_j = \rho_i. \tag{6}$$

Consider the schedule delivered by algorithm *Expand* given $\mathcal{I}$ and ω. Then, for each pair j and k with $j \prec k$, j is scheduled before k provided one of the following three conditions holds: (a) $j \in A$ and $k \in A$, (b) $j \in B$ and $k \in B$, or (c) $j \in A$ and $k \in B$.

Proof: By Lemma 4, since $\mathcal{I}$ satisfies property (1), it is sufficient to show that, in each case, job k is assigned to an interval set at least as late as job j's. For case (a), this is trivial, since j and k are assigned to an interval by *Expand* based entirely on ω, and since Σ is consistent with ω, this implies that k cannot be assigned to an earlier interval than j. Next consider case (b). Since each set J_i is chosen greedily according to availability and delivery time, property (1) guarantees that j will be assigned before k. Finally, consider case (c) and suppose that $j \in I_i$. Clearly, j is assigned to the ith interval, and since properties (1) and (6) imply that $r_k \geq r_j = \rho_i$, job k is certainly assigned no earlier. ∎

In order for this lemma to be useful, we must show that property (6) holds after running procedure *Update-by-release-dates*. This is established by the following lemma.

Lemma 6 If there exists a feasible schedule Σ for an instance $\mathcal{I}$ that is consistent with the given outline ω, procedure *Update-by-release-dates* does not change the release date of any job in $A = \bigcup_{i=1}^{\kappa} I_i$ during step 2.

Proof: Observe that since Σ is consistent with ω, Σ is still feasible with respect to the modified data after step 1 of procedure *Update-by-release-dates*. Furthermore, since the LLR procedure ensures that each feasible schedule remains feasible, Σ is still feasible after step 2. However, each job $j \in I_i$ begins processing in Σ before ρ_{i+1}, and so if the release date of any job in A had been modified in step 2, Σ would no longer be feasible. ∎

Therefore, if we run procedure *Update-by-release-dates* and then algorithm *Expand*, the resulting schedule violates the precedence ordering only for job pairs $j \prec k$ with $j \in B$ and $k \in A$. Suppose that $j \prec k$, $j \in B$, and $k \in I_i$. Since the total amount of small-job-time assigned to $\bigcup_{l=1}^{i} J_l$ is restricted, there is no guarantee that job j will be greedily chosen for one of these sets, particularly if q_k and q_j are very small. We would like to modify q_k so that it accurately reflects the interval to which job k is assigned by the outline; once this is done, condition (1) should be sufficient to ensure that job j has the priority it needs to be chosen for one of the sets $J_1, \ldots, J_i$.

We will first focus on obtaining something slightly weaker than a $(1 + \epsilon)$-approximate outline scheme. Suppose that the preprocessing procedure and algorithm *Expand* are given not only the

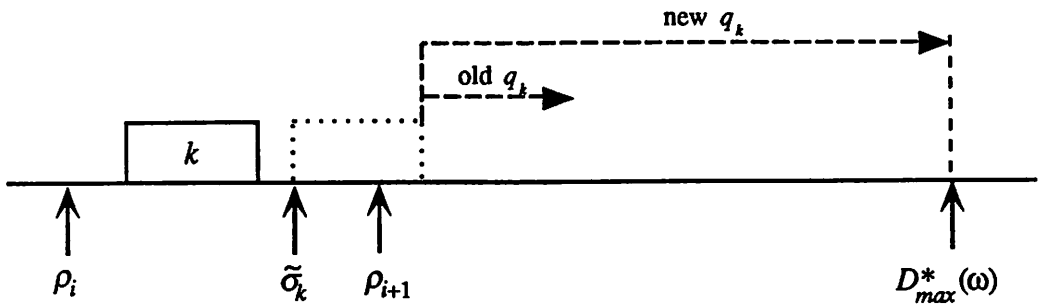


Figure 2: Using the outline to update q_k for $k \in I_i$.

outline ω, but also $D^*_{max}(\omega)$. Consider a job $k \in I_i$, and let $\tilde{\sigma}_k$ be an upper bound on the starting time of k in Σ, which is implied by the outline ω. Then, if a job k has a short delivery time and is forced (by the outline) to be scheduled relatively early in the schedule, the delivery time could be increased to $D^*_{max}(\omega) - (\tilde{\sigma}_k + p_k)$ (see Figure 2). Once this change is made for k, it is natural to further update q_j for each job $j \prec k$.

Of course, it is unreasonable to assume that $D^*_{max}(\omega)$ is given as a part of the input. Instead, we will focus on an algorithm that receives an additional input D, and outputs a schedule Σ' such that the following property holds:

$$D \geq D^*_{max}(\omega) \Rightarrow D_{max}(\Sigma') \leq D + 2\epsilon\Phi \leq D + 2\epsilon D^*_{max}(\omega). \tag{7}$$

The next lemma shows that such a procedure would be sufficient to produce a $(1+2\epsilon)$-approximate outline scheme.

Lemma 7 Suppose that *Modified-Expand* is a polynomial-time algorithm which, given an instance of $1|r_j, prec|L_{max}$ along with an outline ω and a threshold D, either outputs a feasible schedule Σ' with $D_{max}(\Sigma') \leq D + \epsilon' D^*_{max}(\omega)$, or else produces a proof that $D < D^*_{max}(\omega)$. Then this algorithm can be used to construct a $(1+\epsilon')$-approximate outline scheme for $1|r_j, prec|L_{max}$.

Proof: We will embed the algorithm *Modified-Expand* within a binary search procedure. If we consider any canonical schedule Σ, it is straightforward to argue that $D_{max}(\Sigma) \leq \Delta := \max_j r_j + \sum_j p_j + \max_j q_j$ and so we can always obtain a schedule of length no worse than Δ. Thus we can use $UB := \Delta$ and $LB := 0$ as initial upper and lower bounds, respectively, for the binary search; in each iteration, we run *Modified-Expand* with a new midpoint $D := \lfloor (LB + UB)/2 \rfloor$; if there is a schedule output, then update the upper bound to D, otherwise update the lower bound to $D + 1$. The algorithm terminates when $LB = UB$, and outputs the best schedule found. A straightforward proof by induction shows that, throughout the execution of the binary search, the inequality $LB \leq D^*_{max}(\omega)$ always holds, and the best schedule found thus far always has length no more than $UB + \epsilon' D^*_{max}$. After $\log_2 \Delta$ iterations the search converges, and we have obtained a schedule Σ' with

$$D_{max}(\Sigma') \leq (1+\epsilon') D^*_{max}(\omega). \quad \blacksquare$$

At first glance, property (7) appears to be weaker than the one used in Lemma 7; if $D \geq D^*_{max}$, the algorithm works as expected, but how do we tell if this inequality holds? Suppose that the algorithm satisfies (7); then the fact that the schedule Σ' does not satisfy $D_{max}(\Sigma') \leq D + 2\epsilon\Phi$

constitutes a proof that $D < D^*_{max}(\omega)$. Note that it is straightforward to check whether Σ' meets this bound.

Thus, our plan of attack in constructing a polynomial approximation scheme is to devise a preprocessing algorithm that makes use of the parameter D which is now input along with the outline. The algorithm *Modified-Expand* will consist of a preprocessing phase followed by the algorithm *Expand* and will have the following properties:

(i) this preprocessing will ensure that when algorithm *Expand* is run on the modified problem, the precedence constraints are automatically enforced;

(ii) the preprocessing will increase the length of each feasible schedule (consistent with the outline) to at most $D + \epsilon\Phi$ (unlike the preprocessing algorithms discussed above, which leave the objective value unchanged).

We will show that these two properties imply property (7), and therefore such an algorithm would suffice to complete the proof. We must show that if $D \geq D^*_{max}(\omega)$, then *Modified-Expand* outputs a feasible schedule of length at most $D + 2\epsilon\Phi$. Let Σ be a schedule consistent with ω that has $D_{max}(\Sigma) = D^*_{max}(\omega)$. By (ii), the length of Σ after preprocessing increases to at most $D + \epsilon\Phi$. Therefore, by inequality (2), algorithm *Expand* will deliver a schedule Σ' of length at most $(D + \epsilon\Phi) + \epsilon\Phi = D + 2\epsilon\Phi$. Furthermore, by (i), Σ' satisfies the precedence constraints.

As indicated earlier, the ultimate preprocessing algorithm has two stages, and the first of these is procedure *Update-by-release-dates*. The second stage, which we shall call procedure *Update-by-delivery-times*, works as follows:

1. For each job $j \in I_i$, set

$$q_j := \max\{q_j,\ D - (\overline{\rho}'_i + \sum_{k \in I_i} p_k + N_i) + 2\delta\kappa\}. \tag{8}$$

2. Run the LLR procedure.

First we show that the procedure *Update-by-delivery-times* increases the length any schedule Σ consistent with the outline to at most $D + \epsilon\Phi$. For $j \in I_i$, it follows from Lemma 3 that

$$q_j \leq D - (\overline{\rho}'_i + \sum_{k \in I_i} p_k + N_i) + 2\delta\kappa \leq D - (\overline{\rho}_i + \sum_{k \in I_i} p_k) - N_i + \epsilon\Phi$$

and thus job j has been delivered by time

$$D_j \leq \overline{\rho}_i + \sum_{k \in I_i} p_k + M_i + q_j \leq D + (M_i - N_i) + \epsilon\Phi \leq D + \epsilon\Phi.$$

Next we need to show that the precedence constraints are enforced by algorithm *Expand*, under the new delivery times. Since the procedure *Update-by-delivery-times* does not change any release date, Lemma 5 implies that we only need to focus on those pairs of jobs j, k for which $j \in B$, $k \in I_i$, and $j \prec k$. Observe that these conditions clearly imply that such a job j is contained in $\bigcup_{l=1}^{i} H_l$. We wish to show that j must also be in $\bigcup_{l=1}^{i} J_l$. The following lemma will be central to the proof of this result. It states that each small job m which might possibly be selected by *Expand* in advance of job j has also been scheduled in $\bigcup_{l=1}^{i} H_l$, in schedule Σ; that is, in the *original* schedule, job m is scheduled to begin processing before time ρ_{i+1}. This fact will enable us to prove that, in the outline, the values N_i, $i = 1, \ldots, \kappa$, are sufficiently large to allow each small job to be scheduled sufficiently early.

Lemma 8 Given an outline ω for an instance $\mathcal{I}$, let Σ be a feasible schedule consistent with ω such that $D_{max}(\Sigma) = D^*_{max}(\omega)$. Consider jobs j and k with $j \in B$, $k \in I_i$, and $j \prec k$. Then, after executing *Update-by-release-dates* and *Update-by-delivery-dates* along with an input $D \geq D^*_{max}(\omega)$,

$$(m \in B \text{ and } q_m \geq q_j) \Rightarrow m \in H_1 \cup \cdots \cup H_i. \tag{9}$$

Proof: Let $\sigma_1, \ldots, \sigma_n$, respectively, denote the starting times of jobs $1, \ldots, n$ in Σ, and let jobs j and k be as defined in the statement of the lemma. Note that throughout this proof, for each job h, $h = 1, \ldots, n$, r_h and q_h denote the release date and delivery time *after* all preprocessing. Hence, we know that $q_j \geq q_k$. Using the facts that $\overline{p}_i \geq \overline{p}'_i - 2\delta(\kappa - 1)$ and $M_i \geq N_i - \delta$, we see that

$$\begin{aligned} q_j \geq q_k &\geq D - (\overline{p}'_i + \sum_{h \in I_i} p_h + N_i) + 2\delta\kappa \\ &> D - ((\overline{p}'_i - 2\delta(\kappa - 1)) + \sum_{h \in I_i} p_h + (N_i - \delta)) \\ &\geq D^*_{max}(\omega) - (\overline{p}_i + \sum_{h \in I_i} p_h + M_i) \\ &= D^*_{max}(\omega) - (\overline{p}_i + \sum_{h \in I_i \cup H_i} p_h). \end{aligned}$$

Now, consider a job $m \in B$ and suppose that it had delivery time at least q_j *before* applying *Update-by-delivery-times*. Since with this data, the job was delivered by $D^*_{max}(\omega)$, we see that

$$\sigma_m \leq D^*_{max}(\omega) - q_j - p_m = \overline{p}_i + \sum_{h \in I_i \cup H_i} p_h - p_m < \overline{p}_{i+1},$$

which implies that $m \in \bigcup_{l=1}^{i} H_l$.

Next suppose that job m had its delivery time modified to be at least q_j by *Update-by-delivery-times*. Note that for each job $h \in I_i$ that has its delivery time modified in step 1 of *Update-by-delivery-times*, the new delivery time assigned in that step is identical. This implies that by executing *Update-by-delivery-times*, each delivery time is either unchanged, or else equal to one of these κ special values. ¿From this, it follows that job m will have its delivery time modified to be at least q_j only if there exists some job $h \in \bigcup_{l=1}^{i} I_l$ such that $m \prec h$. Since Σ is a feasible schedule consistent with ω, $m \in \bigcup_{l=1}^{i} H_l$.

Therefore, all jobs $m \in B$ with modified delivery times $q_m \geq q_j$ are contained in $\bigcup_{l=1}^{i} H_l$, and the lemma is proved. ∎

This lemma has the following easy corollary.

Corollary 1 Let $\mathcal{I}$ be an instance with a feasible schedule consistent with the outline ω, and consider jobs j and k with $j \in B$, $k \in I_i$, and $j \prec k$. Then, given input $\mathcal{I}$ and ω along with $D \geq D^*_{max}(\omega)$, algorithm *Modified-Expand* schedules j before k.

Proof: We shall show that after executing *Update-by-release-dates* and *Update-by-delivery-times*, algorithm *Expand* will place j into $\bigcup_{l=1}^{i} J_l$. Let r_h and q_h denote the release date and delivery time of job h, $h = 1, \ldots, n$, after preprocessing. If we set $q = q_j$ and apply Lemma 2, we have

$$\sum_{l=1}^{i} \sum_{h \in H_l} (p_h : q_h \geq q_j) \leq \sum_{l=1}^{i} \sum_{h \in J_l} (p_h : q_h \geq q_j).$$

However, by Lemma 8, all jobs in B with delivery time at least q_j are contained in $\bigcup_{l=1}^{i} H_l$. Thus, we see that all of these jobs must also be in $\bigcup_{l=1}^{i} J_l$. In particular, $j \in J_1 \cup \cdots \cup J_i$. ∎

Summarizing the previous discussion, we obtain the following theorem.

Theorem 2 Given an instance $\mathcal{I}$, an outline ω and a threshold D, the algorithm *Modified-Expand* either outputs a feasible schedule Σ' with

$$D_{max}(\Sigma') \leq D + 2\epsilon D^*_{max}(\omega)$$

or else provides a proof that $D < D^*_{max}(\omega)$.

By combining this result with Lemma 7, we obtain a $(1 + 2\epsilon)$-approximate outline scheme, and therefore we have given a polynomial approximation scheme for $1|r_j, prec|L_{max}$.

References

[1] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[2] L. A. Hall and D. B. Shmoys. Jackson's rule for single-machine scheduling: Making a good heuristic better. *Math. Oper. Res.*, to appear.

[3] L. A. Hall and D. B. Shmoys. Approximation algorithms for constrained scheduling problems. In *Proc. IEEE 30th Annual Symp. Foundations of Computer Science*, 134–139, 1989.

[4] J. R. Jackson. *Scheduling a Production Line to Minimize Maximum Tardiness.* Research Report 43, Management Science Research Project, UCLA, 1955.

[5] B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan. Minimizing maximum lateness on one machine: Computational experience and some applications. *Statist. Neerlandica*, 30:25–41, 1976.

[6] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26:22–35, 1978.

On the Impossibility of Strongly Polynomial Algorithms for the Allocation Problem and Its Extensions

Dorit S. Hochbaum[*]
School of Business Administration and IEOR Department
University of California, Berkeley

ABSTRACT

The allocation problem is to find the maximum of a separable concave function over a cardinality constraint that specifies the sum of the integer variables. This problem is solvable using a greedy algorithm in pseudo-polynomial time that depends on the cardinality. The constraint set can be generalized so the property of solvability via greedy is maintained. The most general form is maintaining this property is when the constraints polytope is a polymatroid.

We present a scaling-based algorithm that provides polynomial algorithms for these problems. The scaling (along with a proximity result developed for this case) reduces the problem to a problem with small right-hand-sides which is solvable, by greedy, in (strongly) polynomial time. These algorithms also apply for problems with continuous variables.

The algorithms resulting from the scaling approach are the fastest known for these problems. These algorithms however are not strongly polynomial. We prove a lower bound result in the algebraic tree model (four arithmetic operations and comparisons) that establishes that strongly polynomial algorithms are impossible for the allocation problem and its extensions.

1. Introduction and Summary

The major results in this paper include a proof of impossibility of strongly polymomial algorithms for optimizing a nonlinear function over linear polytope even when the polytope is defined by a simple totally unimodular matrix; the description and validation of an algorithmic technique based on scaling and proximity for the general allocation problem and finally a collection of algorithms that are optimal and are all based on scaling and proximity.

We consider in this paper the separable concave allocation problem, $\text{Max}\{\sum_{i=1}^{n} f_i(x_i) \mid \sum_{i=1}^{n} x_i = B,\ x \geq 0\}$ and its extensions to problems with x satisfying in addition general

(*) Supported in part by the Office of Naval Research under grant N00014-88-K-0377.

polymatroidal constraints. We call such problems, *general allocation problems*. These problems are studied for both cases when either x is an integer or continuous vector. Other than the concavity of the f_is, no additional assumptions are made.

The general allocation problem is a special case of nonlinear problems over linear polytopes. As such it follows from a recent paper [HS88], that both the integer and continuous versions of this problem are solvable in time polynomial in the input length and the value of maximum subdeterminant of the constraint matrix. The idea in [HS88] is to scale the problem on successively refined grids. The granularity of the grid depends on a proximity theorem that guarantees the closeness of the scaled solution to the optimal solution. This closeness is a linear functoin of the number of variables and the value of the maximum subdeterminant. Consequently, it implies directly apolynomial algorithm for the simple resouce allocation problem and some of its extensions.

All the integer versions of the general allocation problems are characterized as solvable via a greedy allocation procedure. The value of each variable is incremented by one unit at a time if the corresponding increase in the objective function is largest among all possible increments that are feasible, until all B units are allocated. This greedy algorithm was first devised by Gross [G56] and later by Fox [F65] for the integer allocation problem. The difficulty with this algorithm is that it requires O(B) iterative steps, each including at least one evaluation of the objective function. Since the representation of B in the input takes only $\log_2 B$ bits, the running time of the greedy is exponential (or pseudo-polynomial) in the length of B, and hence exponential in the input length.

In order to fully characterize the complexity of the greedy algorithm , one needs to know the form of the representation of the objective function, and the complexity of evaluating those functions' values. Since the functions f_i are assumed to be general, we assume the existence of an oracle computing the functions' values. An oracle call is counted as a single operation in the complexity model employed in this paper. With this complexity model, the running time of the greedy algorithm for the general allocation problem is O(B[logn+F]), where F is the number of operations required to check the feasibility of a given increment in the solution.

The results given in this paper establish constructively, that the general integer allocation problem is solvable in **polynomial** time, requiring $O(\log_2 \frac{B}{n})$ iterations. This polynomial algorithm is developed using concepts of scaling and proximity between the optimal solutions of two scaled problems. Scaling and proximity also are the main ingredients making the polynomial algorithm for general nonlinear programming problems work ([HS88]). Here however the proximity is stronger and the linear problem that results from the reduction is solved very efficiently using the greedy algorithm. The running time of our algorithm for the general integer allocation

problem is $O(n(\log n+F)\log_2\frac{B}{n})$), and for the continuous case an ε - accurate solution (to be defined subsequently) is produced in $O(n(\log n+F)\log_2\frac{B}{\varepsilon n})$. Moreover, we show that the dependence on $\log\frac{B}{\varepsilon}$ in the running time of the algorithm cannot be removed even for the relatively simple case of the allocation problem. This lower bound result holds either in the comparison model or in the algebraic computation tree model with the four arithmetic operations $+,-,\cdot,:$, and comparisons (and the floor operation). Since the allocation problem is a special case of separable concave maximization problems over totally unimodular constraints matrices or over polymatroidal constraints, this problem and the general nonlinear optimization problem cannot be solved in strongly polynomial time.

This issue of strong polynomiality is of special interest, as it clearly delineates the distinction in the complexity of nonlinear versus that of linear problems. With linear objective function the integer programming problem over totally unimodular constraint matrix is solvable in strongly polynomial time, ([T86]), whereas the separable concave version of this problem is solvable in time which depends on the logarithm of the right hand side ([HS88]). With the lower bound result given here, the conclusion is that this dependence on the right hand side elements cannot be removed.

The allocation problem and its extensions have been studied extensively. A recently published book by Ibaraki and Katoh [IK88], gives a complete review of the state of the art of the problem, as well as more than 150 references. Our results here improve on the algorithms for the general allocation (integer) problem and its special cases as well as provide polynomial algorithms for the problem in **continuous** variables. The continuous general allocation problem has only been solved in special cases when the derivatives of the f_is exist. For example the ranking algorithm suggested by Zipkin, [Z80] for the allocation problem, not only requires the existence of the derivatives, but also requires the solution of a system of nonlinear equations arising of these derivatives. Solving a system of nonlinear equations is a challenging problem even when the nonlinearities are as simple as polynomials. Similar difficulties are encountered in the algorithms proposed by [YS87]. In contrast, we do not require here the existence of derivatives.

It is not surprising that these difficulties are encountered in the continuous version of the (general) allocation problem since the solution could require infinite representation (that exists in the example above). For example, let a simple allocation problem be given with $f_1(x_1)=6x_1-x_1^3$, $f_2(x_2)=0$, and $B=2$. The optimal solution to the 2 variable allocation problem is $(\sqrt{2}, 2-\sqrt{2})$, which is irrational. The solution may even lack an algebraic representation. We therefore use the practical notion of ε - *accurate* solution to solve the problem. A solution, $x^{(\varepsilon)}$ is ε - *accurate* if there exists an optimal solution x^* such that $\| x^{(\varepsilon)}-x^* \|_\infty \le \varepsilon$. That is, ε is the accuracy required of

the solution produced.

Using the proximity results, we show that ε - accurate solutions can be obtained by solving a general integer allocation problem obtained by scaling the continuous general allocation problem by a factor of $\frac{\varepsilon}{n}$. Hence, the continuous problem is reduced to the integer case, and similar algorithms to those used in the integer case apply to the continuous case. Furthermore, the lower bound results establish that the dependence on $\frac{B}{\varepsilon}$ cannot be removed, even in the continuous case.

2. Preliminaries and Notation

Given a *submodular* rank function $r:2^E \rightarrow R$, for $E = \{1, ..., n\}$, i.e. $\forall A, B \supseteq E$,

$$r(A) + r(B) \geq r(A \cup B) + r(A \cap B)$$

(for more detailed definition see e.g. [NW88]). The *polymatroid* defined by the rank function r, is the polytope $\{x \mid \sum_{j \in A} x_j \leq r(A), \quad \forall A \subseteq E\}$.

We call the system of inequalities

$$\sum_{j \in A} x_j \leq r(A) \quad \forall A \subseteq E,$$

the *polymatroidal constraints.*

Regarding the notation in this paper, bold letters are used to denote vectors; **e** is the vector $(1, 1, ..., 1)$; $\mathbf{e^j}$ is the unit vector with $e^j_j = 1$ and $e^j_i = 0$ for $i \neq j$; all logarithms are base 2 logarithms. The general allocation problem, GAP, is then

$$\text{(GAP)} \qquad \max \sum_{j \in \epsilon} f_j(x_j)$$

$$\sum_{j \in E} x_j = r(E)$$

$$\sum_{j \in A} x_j \leq r(A) \quad \forall\, A \subset E$$

$$x_j \geq l_j \text{ and integer } \forall\, j \in E.$$

We give the problem in a slightly generalized form with the lower bound l_j rather than nonnegativity requirements as is common in the literature. It is well known that the greedy algorithm solves the problem (GAP) (e.g., [FG86], [IK88]). The concept of *increment* is used in the greedy: the j^{th} increment at x_j is:

$$\Delta_j(x_j) = f_j(x_j + 1) - f_j(x_j).$$

We now describe formally the greedy for GAP.

Procedure greedy

Input (l, r, E)

Step 0: $x = l$, $B = r(E) - l \cdot e$.

Step 1: Find i such that $\Delta_i(x_i) = \max_{j \in E} \{\Delta_j(x_j)\}$.

Step 2: (feasibility check), Is $x + e^i$ infeasible?
If yes, $E \leftarrow E - \{i\}$,
else $x_i \leftarrow x_i + 1$
$B \leftarrow B - 1$.

Step 3: If $B = 0$ stop. Output x. If $E = \Phi$, stop, output "no feasible solution". Otherwise go to step 1.

Note that the feasibility check is polynomial for the general allocation problem (using the ellipsoid method, [GLS81]), and can be performed trivially in a single step for the simple allocation problem. Note also that the output of "no feasible solution" in Step 3 never materializes if r is

indeed a submodular function, and the choice of the vector 1 is such that it does not violate the polymatroidal constraints (that is 1 is a member of the polymatroid).

The *greedy* algorithm is applicable to special cases of GAP. Important special cases that have been studied in the literature are: the simple resource allocation pronlem, the generalized upper bound. resource allocation , the nested and the tree constrained resource allocation problem.

3. Lower Bounds

In contrast to Linear programming where for a maximum subdeterminant of the constraint matrix of polynomial length, there is a strongly polynomial algorithm to solve the problem, (see Tardos [T86]), this is not the case when we allow to introduce a concave objective function, even if it is separable and consists only of two (or one) variables, and even if the maximum subdeterminant is 1, (i.e. the constraint matrix is totally unimodular), and even if there is only one constraint in addition to nonnegativity.

3.1 A comparison Model Lower Bound

We provide a lower bound proof that establishes that no algorithm exists that can solve SRA in less than $\log_2 B$ comparisons. In this, we rely on a result of information theory according to which there is no algorithm that finds a value in a monotonic decreasing n-array that is the first to be smaller than some specified constant in less than $\log_2 n$ time.

The comparison model the allocation problem on $n+1$ variables has complexity $\Omega(n \log_2 \frac{B}{n})$. Let the problem be defined for $c > 0$,

$$\sum_{j=1}^{n} f_j(x_j) + c \cdot x_{n+1}$$

$$\sum_{j=1}^{n+1} x_j = B$$

$$x_j \geq 0 \text{ and integer } j = 1, \ldots, n+1 .$$

Let the functions f_j be concave and monotonic increasing in the interval $[0, \lfloor \frac{B}{n} \rfloor]$, and zero in $[\lfloor \frac{B}{n} \rfloor, B]$. Solving this problem is then equivalent to determining in n arrays of length $\lfloor \frac{B}{n} \rfloor$

each, the last entry of value $\geq c$. Since the arrays are independent, the information theory lower bound is $\Omega(n \log\lfloor \frac{B}{n} \rfloor)$. Similarly, for the case of an inequality constraint the same lower bound supplies for the problem on n variables

$$\max \sum_{j=1}^{n} f_j(x_j)$$

$$\sum_{j=1}^{n} x_j \leq B$$

$$x_j \geq 0 \quad \text{integer } j = 1 \ldots n$$

since x_{n+1} can simply be viewed as the slack and $c = 0$.

3.2 The Algebraic-Tree Model

Due to the nature of the problem one might criticize the choice of the comparison model for this problem. Indeed, the use of arithmetic operations may help to reduce the problems complexity. The computation model that is used hereafter allows the arithmetic operations $+, -, \times, \div$ as well as comparisons and branching based on any of these operations. Here we rely on Renegar's lower bound proof ([R87]) in this arithmetic model of computation for finding ε-accurate roots of polynomials of fixed degree ≥ 2 polynomials. In particular, the complexity of identifying an ε-accurate single real root in an interval $[O, R]$ is $\Omega(\log\log \frac{R}{\varepsilon})$ even if the polynomial is monotonic in that interval.

Let $p_1(x), \ldots, p_n(x)$ be n polynomials each with a single root to the equation $p_i(x) = c$ in the interval $[0, \frac{B}{n}]$, and each $p_i(x)$ is a monotonic decreasing function in this interval. Since the choice of these polynomials is arbitrary, the lower bound on finding the n roots of these n polynomials is $\Omega(n \log\log \frac{B}{n\varepsilon})$. Let $f_j(x_j) = \int_0^{x_j} p_j(x)\,dx$. The f_j s are then polynomials of degree ≥ 3. The problem,

$$(P_\varepsilon) \qquad \max \sum_j f_j(x_j \cdot \varepsilon) + c \cdot x_{n-1} \cdot \varepsilon$$

$$\sum_{j=1}^{n+1} x_j = \frac{B}{\varepsilon}$$

$$x_j \geq 0$$

$$x_j \text{ integer},$$

has an optimal solution x such that $y = \varepsilon \cdot x$ is the $(n\varepsilon)$-accurate vector of roots solving the system

$$\begin{cases} p_1(y_1) = c \\ p_2(y_2) = c \\ \vdots \\ p_n(y_n) = c. \end{cases}$$

This follows directly from the Kuhn-Tucker conditions of optimality, and the fact that an optimal integer solution to the scaled problem with a scaling constraint, s, x^*, and the optimal solution to the continuous problem y^* satisfy $\| x^* - y^* \|_\infty \leq ns$. (This proximity result was proved in [HS88]). Hence a lower bound for the complexity of solving (P_ε) is $\Omega(n \log\log \frac{B}{n^2\varepsilon})$.

It is important to note that in [MST88] there is a lower bound proof for finding ε-accurate square root that allows also the floor, $\lfloor \ \rfloor$, operation. In our terminology the resulting lower bound for our problem is $\Omega(\sqrt{\log \frac{B}{\varepsilon}})$, hence even with this additional operation the problem cannot be solved in strongly polynomial time. Again, the quadratic objective is an exception and the algorithms for solving the quadratic objective resource allocation problems rely on solving for the continuous solution first, then rounding down, (using the floor operation), and proceeding to compute the resulting integer vector to feasibility and optimality using fewer than n greedy steps. Since the lower bound result applies also in the presence of the floor operation, it follows that the "ease" of solving the quadratic case is indeed due to the quadratic objective and not to this, perhaps powerful, operation.

4. A Proximity Theorem

Consider the scaled problem, GAPs,

$$\text{(GAPs)} \qquad \max \sum_{j\in E} f_j(sx_j)$$

$$\sum_{j\in E} x_j = \frac{r(E)}{s}$$

$$\sum_{j\in A} x_j \le \frac{r(A)}{s} \qquad \forall\, A \subset E$$

$$x_j \ge \frac{l_j}{s} \text{ and integer } \forall\, j \in E.$$

Like (GAP), (GAPs) is also solvable by a scaled analogue of the greedy derived by solving, using the unscaled right hand sides, replacing Δ_j by $\Delta_j^{(s)}(x_j) = f_j(x_j + s) - f_j(x_j)$, and incrementing the selected component by s units.

A direct application of the algorithms in [HS88] indeed calls for finding an optimal integer solution to this scaled problem. Here we employ however a different approach that allows for tighter proximity and faster running time. We use an algorithm - *greedy(s)* - that compares the increments Δ_j as in *greedy*, (rather than $\Delta_j^{(s)}$) but the increase of the selected component is s units, when an increase of only **one** unit is feasible.

Procedure greedy(s)

Step 0: $x = l$, $B = r(E) - l \cdot e$, $E = \{1, 2, ..., n\}$.

Step 1: Find i such that $\Delta_i(x_i) = \max_{j\in E} \{\Delta_j(x_j)\}$.

Step 2: (feasibility check), Is $x + e^i$ infeasible?
If yes, $E \leftarrow E - \{i\}$,
else, $x_i \leftarrow x_i + s$
$B \leftarrow B - s$.

Step 3: If $B \le 0$, stop, output x.
(If $E = \Phi$, stop output "no feasible solution".) Otherwise go to step 1.

Let the output of *greedy(s)* be $x^{(s)}$, and the output of *greedy* which is an optimal solution to (GAP), x^*. Note that $x^{(s)}$ is an integer multiple of s, and it may violate the same constraint of GAPs by $s-1$ units at most.

Proximity Theorem (Theorem 4.1). $\exists x^*$ such that $x^* \geq x^{(s)} - s \cdot e$.

Discussion: This theorem is precisely of the same flavor as the one that has been used by Edmonds and Karp [EK72] to solve the minimum cost network flow problem via scaling. The differences are that in [EK72] there is no truncation of the scaled solution, and that here the optimal solution to the scaled problem is not necessarily an integer multiple of s.

5. The Main Algorithm

Given a general resource allocation problem, GAP, with 1 a feasible solution and $r(E) = B$. The following algorithm that is based on the idea of scaling and proximity solves the GAP problem in $O(n(\log n + F)\log \frac{B}{n})$ operations, with F denoting the running time (in *greedy* or *greedy(s)*) required to verify that an increment in one of the vector's component is feasible. Note that *greedy* is identified with *greedy(1)*.

Algorithm GAP

Step 0: Let $s = \lceil \frac{B}{2n} \rceil$.

Step 1: If $s = 1$, call *greedy*. The output is x^*. Stop, x^* is an optimal solution.

Step 2: Call *greedy(s)*. Let $x^{(s)}$ be the output.
Set $1 = x^{(s)} - se$
Set $s \leftarrow \lceil \frac{s}{2} \rceil$.
Go to step 1.

Theorem 5.1.

(a) Algorithm GAP is valid;

(b) The running time of Algorithm GAP is $O(n(\log n + F)\log \frac{B}{n})$.

Proof:

(a) The validity is a direct corollary of the proximity theorem. Since the vector $x^{(s)} - s \cdot e$ bounds the optimum from below, setting the lower bound constraints to $x \geq x^{(s)} - s \cdot e$ does not change the solution to GAP.

(b) *greedy(s)* is called $\log\lceil \frac{B}{2n} \rceil$ times and *greedy* is called once. Each time a call is made there are at most $\lceil \frac{B - l \cdot e}{s} \rceil$ increments to be executed.

$$l \cdot e = \sum_{i=1}^{n} l_i = \sum_{i=1}^{n} \left\lfloor x_i^{(s)} - s \right\rfloor = \sum_{i=1}^{n} x_i^{(s)} - sn \geq B - sn .$$

Hence, at the next iteration $\lceil \frac{B - l \cdot e}{s/2} \rceil \leq 2n$. Thus, there are no more than O(n) increments at each call to be executed by *greedy(s)*, or *greedy*. The amount of work required for each increment is $O(\log_2 n + F)$, where $O(\log_2 n)$ comparisons are needed to maintain the sorted vector of up to n potential increments and F steps to check the feasibility of a selected increment. Note that if an increment in component j is not feasible, that component is removed from further consideration. Consequently, there are at most n such "failed" feasibility tests. □

For the special cases of the allocation problems there are more efficient implementations that are optimal. The description of these implementations is given in the full version of the paper.

REFERENCES

[B84] P. Brucker, "An O(n) Algorithm for Quadratic Knapsack Problems," *Operations Research Letters*, Vol. 3 No. 3 (1984), 163-166.

[EK72] J. Edmonds and R. M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," *J. of ACM*, 19 (1972), 248-264.

[FG86] A. Federgruen and H. Groenvelt, "The Greedy Procedure for Resource Allocation Problems: Necessary and Sufficient Conditions for Optimality," *Operations Research*, 34 (1986), 909-918.

[F66] B. L. Fox, "Discrete Optimization via Marginal Analysis," *Management Science,* 13, (1966) 210-216.

[FJ82] G. N. Frederickson and D. B. Johnson, "The Complexity of Selection and Ranking in X + Y and Matrices with Sorted Columns," *J. of Computer and Systems Sciences,* 24 (1982), 197-208.

[GT85] H. N. Gabow and R. E. Tarjan, "A Linear-Time Algorithm for a Special Case of Disjoint Set Union," *J. of Computer and System Science,* 30 (1985), 209-221.

[GLS81] M. Grotchel, L. Lova´sz and A. Schrijver, "The Ellipsoid Method and its Consequences in Combinatorial Optimization," *Combinatorica,* 1 (1981), 169-197.

[G56] O. Gross, "A Class of Discrete Type Minimization Problems," RM-1644, Rand Core. (1956).

[HS88] D. S. Hochbaum and J. G. Shanthikumar, "Convex Separable Optimization is Not Much Harder than Linear Optimization," U.C. Berkeley, manuscript, April (1988), to appear, *J. of ACM.*

[IK88] T. Ibaraki and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches,* MIT Press (1988).

[K73] D. E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching,* Addison Wesley, (1973) 183-184.

[MST88] Y. Mansour, B. Schieber and P. Tiwari, "Lower Bounds for Computations with the Floor Operations," manuscript, December (1988).

[NW88] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization,* Wiley (1988).

[R87] J. Renegar "On the Worst Case Arithmetic Complexity of Approximation Zeroes of Polynomials," *J. of Complexity,* 3, (1987) 9-113.

[T86] E. Tardos, "A Strongly Polynomial Algorithm to Solve Combinatorial Linear Programs," *Operations Research,* 34, (1986) 250-256.

[YS87] D. D. Yao and J. G. Shanthikumar, "The Optimal Input Rates to a System of Manufacturing Cells," *INFOR*, 25 (1987), 57-65.

[Z80] P. Zipkin, "Simple Ranking Methods for Allocation or one Resource," *Management Science*, 26, (1980) 34-43.

Shellability of Oriented Matroids

Winfried Hochstättler, Cologne

Abstract

In [Man82] A. Mandel proved that the maximal cells of an Oriented Matroid poset are B-shellable. Our result shows that the whole Oriented Matroid is shellable, too.

1 Introduction

In the late seventies Oriented Matroid theory was mainly developed to get more insight into the geometry of the Simplex algorithm. But also in one of the first articles about Oriented Matroids, Folkman and Lawrence [FoLa78] observed a strong correspondence to cell complexes of arrangements of pseudohemispheres known from combinatorial topology. So we can study cell complexes of Sphere Systems by means of signed vectors in Oriented Matroids.

Mandel proved in his thesis [Man82] that the face lattice of an Oriented Matroid - known as Edmonds-Fukuda-Mandel lattice - has a spherical structure, too. To be more precise, he showed that topes (the copoints in the lattice) are shellable. Unfortunately he could not prove shellability of the Oriented Matroid itself. To paste all the topes together, he introduced the slightly weaker concept of constructability. In this paper we will show that this is not necessary in this context, since Oriented Matroids in fact are shellable.

The paper is organized as follows: In section 2 we are facing essentially three longer definitions and in section 3 we cite some known results on tope lattices, so the reader may skip these, if he is familiar with that material, and read only section 4.

2 Basic Concepts

At first we have to introduce the structures we are going to deal with and collect some tools needed later on. We will not go into details, a good reference is the forthcoming book by Bachem, Kern [BaKe90].

Oriented Matroids can be defined in various equivalent ways. We will use the notion of a Complete Oriented Matroid.

Definition 1 *A* **signed vector** U *on a finite set* E *is a map* $U : E \to \{0, +, -\}$. *Instead of* $U(e)$ *we will simply write* U_e. *By* $2^{\pm E}$ *we denote the set of all signed vectors on* E. *If* $A \subseteq E$ *then* $_{\bar{A}}U$ *is the vector derived from reversing signs on* A, *i.e.* $_{\bar{A}}U_e = U_e$ *for all* $e \in E \setminus A$, *and* $_{\bar{A}}U_e = +$ *if and only if* $U_e = -$ *and vice versa for all* $e \in A$. *Of course* $-U := {}_{\bar{E}}U$. *If* U *and* V *are signed vectors* $\mathbf{sep(U, V)} := \{e \in E \mid U_e = -V_e \neq 0\}$ *and* $\mathbf{supp(U)} := U^{-1}(\{+, -\})$. *Furthermore the* **composition** $Z = U \circ V$ *of* U *and* V *is that signed vector* Z *with* $Z_e := U_e$ *for all* $e \in supp(U)$ *and* $Z_e := V_e$ *elsewhere.*

An **Oriented Matroid** $\mathcal{O}$ *on a finite set* E *is a system of signed vectors satisfying*

1. $0 \in \mathcal{O}$ *and* $(U \in \mathcal{O} \Longrightarrow -U \in \mathcal{O})$ *(Symmetry)*
2. $\forall U, V \in \mathcal{O} : U \circ V \in \mathcal{O}$
3. $\forall U, V \in \mathcal{O}, U \neq -V \quad \forall e \in sep(U, V) \quad \exists Z \in \mathcal{O}:$
   $(Z_e = 0 \quad and \quad \forall g \notin sep(U, V) : Z_g = (U \circ V)_g)$

(Approximation Property)

We look at Oriented Matroids as systems of signed vectors, however they may also be considered as Sphere Systems as follows.

Definition 2 *A* **topological n-ball ($n-1$-sphere)** *is a topological space B_n (S_{n-1}), homeomorphic to an n-dimensional unit ball $B^n = \{x \in \mathbb{R}^n \mid \|x\| \leq 1\}$ ($n-1$-dimensional unit sphere $S^{n-1} = \{x \in \mathbb{R}^n \mid \|x\| = 1\}$), $n \geq 0$. Please, note that by convention $\mathbb{R}^0 = \{0\}$.*

If S_n is a topological n-sphere and $h : S^n \to S_n$ is any homeomorphism, then $H^0 = h(S^{n-1} \times \{0\})$ is called a **hypersphere** *of S_n. Marking one of the two connected components of $S_n \setminus H^0$ by " $+$ " and the other one by " $-$ ", we get an* **oriented hypersphere** *(H^0, H^+, H^-) of S. The components H^+ and H^- are called* **sides** *of H^0 in S_n. Furthermore we have two* **closed halfspheres** *$\overline{H^+} := H^0 \cup H^+$ and $\overline{H^-} := H^0 \cup H^-$.*

Now, let $\mathcal{H}$ be a finite collection of oriented hyperspheres of S_n indexed by E. If $A \subseteq E$ then $F := \bigcap_{e \in A} H^0_e$ is called a **flat** *of S. The intersection of any set of closed halfspheres which is not a flat is a* **supercell**. *A supercell P is a* **cell**, *if for any supercells Q, R we have, if $Q \cup R = P$ then either $Q \subseteq R$ or $R \subseteq Q$.*

The triple $(S_n, E, \mathcal{H}) =: S$ is called a **Sphere System** *if,*

1. *Any flat is a topological sphere.*
2. *If F is a flat, H^0_e is a hypersphere of S and $F \not\subseteq H^0_e$ then $H^0_e \cap F$ is a hypersphere of F with sides $F \cap H^+_e$ and $F \cap H^-_e$.*
3. *Any supercell is a topological ball.*

Theorem 1 ([FoLa78]) *Any finite Sphere System gives rise to an Oriented Matroid and vice versa.*

Let us explain this by an example (see fig. 1). Consider $S := (S^2, E, \mathcal{H})$ with $\mathcal{H}$ a finite set of "oriented" great circles indexed by E on S^2 which do not all meet in one point. To each great circle $H^0_e \in \mathcal{H}$ we associate an upper half H^+_e and and a lower half H^-_e of $S^2 \setminus H^0_e$. This gives rise to a map $\sigma : S^2 \to 2^{\pm E}$ $p \mapsto \sigma(p) = U$ with $U_e = j \Leftrightarrow p \in H^j_e$, $j \in \{0, +, -\}$. These, together with 0, form the set of signed vectors of an Oriented Matroid. Besides $\sigma^{-1}(U)$ is a cell of S^2 for any $U \in \sigma(S^2)$. Obviously inclusion of the their closures gives rise to a partial order on the cells of S^2. This carries over to our Oriented Matroid in the following way.

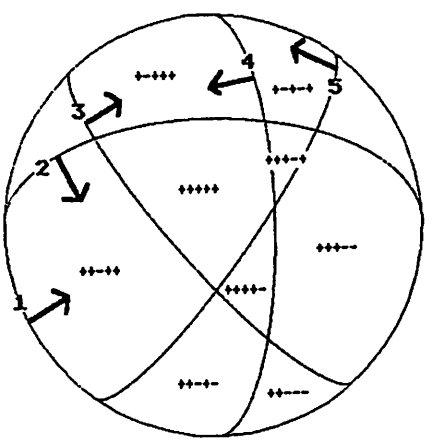


Figure 1: A Sphere System and some maximal signed vectors

Definition 3 *Let $\mathcal{O}$ be an Oriented Matroid on a finite set E and $U, V \in \mathcal{O}$. We say X* **conforms** *to Y and write $X \preceq Y$, if for all $e \in E : X_e \neq Y_e$ only if $X_e = 0$. If $X \preceq Y$, $X \neq Y$ and for all $Z \in \mathcal{O} : X \preceq Z \preceq Y$ only if either $Z = X$ or $Z = Y$ we say Y* **covers** *X or X is a* **facet** *of Y. The maximal elements of this partial order (poset) are called* **topes** *of $\mathcal{O}$.*

Theorem 2 **([Man82])** *The poset $(\mathcal{O}, \preceq)$ is pure d-dimensional for some $d \in \mathbb{N}$, i.e. it has a dimension function and all its maximal elements have the same dimension d. If we add an artificial supremum of all maximal elements of $\mathcal{O}$ resulting in $\bar{\mathcal{O}}$, then $(\bar{\mathcal{O}}, \preceq)$ is a lattice. The intervalls of dimension 1 in this lattice always look like a diamond, i.e. they contain precisely four elements.*

Before we turn to shellability of posets, let us again consider our example. Of course a maximal cell of our complex looks like a 2-dimensional ball and, if we peel it off, so does the remaining complex (see figure 2). Furthermore their intersection is a 1-dimensional sphere. If we regard two adjacent maximal cells, their union also looks like a 2-dimensional ball and their common boundary is a 1-dimensional ball. Summing this up we get the following recursive definition.

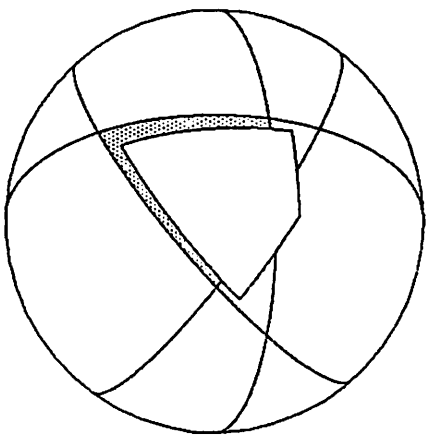


Figure 2: Peeling off a maximal cell

Definition 4 *Let $\mathcal{P} = (P, \leq)$ be a pure d-dimensional poset.*

If $Q \subseteq P$ by $[Q]$ we denote the order ideal induced by Q. We say Q is closed if $Q = [Q]$. If Q is closed, the **boundary** *of Q is $\partial Q := \{q \in P \mid \exists q' \in P : q \leq q'$ and q' is covered by exactly one maximal element in $Q\}$.*

*$\mathcal{P}$ is S-***shellable of dimension** *d if either $d = -1$ or $\mathcal{P} = [P_1] \cup [\{p\}]$, where $[P_1]$ and $[\{p\}]$ are B-shellable of dimension d and $[P_1] \cap [\{p\}] = \partial[P_1] = \partial[\{p\}]$ is S-shellable of dimension $d-1$.*

*$\mathcal{P}$ is B-***shellable of dimension** *d if either $\mathcal{P} = [\{p\}]$ and $\partial[\{p\}]$ is S-shellable of dimension $d-1$ or $\mathcal{P} = [P_1] \cup [\{p\}]$, where $[P_1]$ and $[\{p\}]$ are B-shellable of dimension d and $[P_1] \cap [\{p\}] = \partial[P_1] \cap \partial[\{p\}]$ is B-shellable of dimension $d-1$.*

3 Tools

In this section we will consider the topes of Oriented Matroids. It will make our work much easier forgetting about elements of E that do not add any structure to $\mathcal{O}$.

Definition 5 *Let $\mathcal{O}$ be an Oriented Matroid on a finite set E and $e, f \in \mathcal{O}$. We call e a* **loop**, *if $U_e = 0$ for all $U \in \mathcal{O}$, e and f are* **(anti-) parallel**, *if $U_e = U_f$ $(U_e = -U_f)$ for all $U \in \mathcal{O}$. An Oriented Matroid is* **simple**, *if it has no loops and (anti-) parallels.*

If $\tilde{E} = E \setminus e$, then the set $\mathcal{O} \setminus e := \{\tilde{X} \in 2^{\pm\tilde{E}} \mid \exists X \in \mathcal{O} : (X_e = 0 \text{ and } \tilde{X} = X_{|\tilde{E}})\}$ is an Oriented Matroid again. We say, it is derived by **deletion** *of e. Similarly by* **contraction** *of e we get: $\mathcal{O}/e := \{\hat{X} \in 2^{\pm\tilde{E}} \mid \exists X \in \mathcal{O} : (X_e \text{ arbitrary and } \hat{X} = X_{|\tilde{E}})\}$*

Lemma 1 *If e is a loop in $\mathcal{O}$ and $\tilde{\mathcal{O}} := \mathcal{O} \setminus e$ then $(\tilde{\mathcal{O}}, \preceq)$ and $(\mathcal{O}, \preceq)$ are isomorphic. If e and f are (anti-) parallel and $\hat{\mathcal{O}} := \mathcal{O}/e$, then $(\hat{\mathcal{O}}, \preceq)$ and $(\mathcal{O}, \preceq)$ are isomorphic.*

So in the following we may always assume that $\mathcal{O}$ is simple. This is very helpful, since the topes and their facets are now easy to describe.

Lemma 2 *If $\mathcal{O}$ is simple, T is a tope of $\mathcal{O}$ and Y is a facet of T, then $supp(T) = E$ and there exists an $e \in E$ such that $supp(Y) = E \setminus e$. In this case we call e the* **facet equator** *of Y.*

Let us have a closer look at a tope and its facets. First we want to examine the neighbourhood of one facet and then introduce the socalled umbrellas of a tope which are special connected sets of facets. We will see that proper umbrellas always look like balls, they do not have holes, as it should be with proper umbrellas.

Lemma 3 *Let T denote a tope and $Y \preceq T$ a maximal cell satisfying $Y_e = 0$, then*

$$Y \text{ is a facet of } T \text{ if and only if } {}_eT \text{ is a tope}$$

Definition 6 *Let T denote a tope and $\mathcal{U}$ a set of facets of T. Then $\mathcal{U}$ is an* **umbrella** *of T, if there exists a tope T' of $\mathcal{O}$; such that $\mathcal{U}$ is exactly the set of facets of T with facet equator $e \in sep(T, T')$ (see figure 3). We write $\mathcal{U} = \mathcal{U}(T, T')$. We say $\mathcal{U}$ is a* **proper** *umbrella, if it does not contain all facets of T.*

Lemma 4 *Let $T \neq T'$ be topes of $\mathcal{O}$. Then there exists a facet Y of T with facet equator $e \in sep(T, T')$.*

Lemma 5 *Let $T \neq T'$ be topes of $\mathcal{O}$. Then*

$$\mathcal{U}(T, T') \text{ is a proper umbrella of } T \text{ if and only if } T' \neq -T$$

Before we finish this section, let us include powerful tool. As mentioned above most of our work has already been done. Since topes are B-shellable, we do not have to worry much about lower dimensional cells.

Theorem 3 *Let $\mathcal{O}$ denote an Oriented Matroid of dimension d on a finite set E and T a tope of $\mathcal{O}$. Then the following holds:*

1. *$[T]$ is B-shellable of dimension d*
2. *The closure of every proper umbrella of T is B-shellable of dimension $d-1$*
3. *$\partial[T]$ is S-shellable of dimension $d-1$*

4 Our Result

Let us look at the single steps when shelling an Oriented Matroid. We have to peel off an arbitrary tope $-T$, because there is no special candidat for this, and prove that the remaining complex also looks like a ball. But now, there is a special tope on the other side of our complex, namely T. We will take this as a kernel and let our complex grow, always keeping it looking like a ball. We are going to add one by one that remaining tope which is next to T, whereas the number of elements separating our topes from T serves as a distance function. In other words: We start a breadth-first-search on the tope graph.

Definition 7 *Let $\mathcal{O}$ be an Oriented Matroid of dimension d on a finite set E and $\mathcal{L}$ a set of topes of $\mathcal{O}$. We call $\mathcal{L}$ a* **lump (centered at T of width k)** *if there exists a tope T in $\mathcal{L}$ and a non-negative integer k, such that*

1. $\exists T' \in \mathcal{L} : |sep(T, T')| = k$
2. $\forall$ *topes* $T'' \in \mathcal{O} : (|sep(T, T'')| > k \Longrightarrow T'' \notin \mathcal{L})$
3. $\forall$ *topes* $T'' \in \mathcal{O} : (|sep(T, T'')| < k \Longrightarrow T'' \in \mathcal{L})$

If in addition we have

4. $\forall$ *topes* $T'' \in \mathcal{O} : (|sep(T, T'')| = k \Longrightarrow T'' \in \mathcal{L})$

then $\mathcal{L}$ is a **complete** *lump.*

Note that a single tope is a complete lump of width 0 and the set of all topes is a complete lump of width $|E|$.

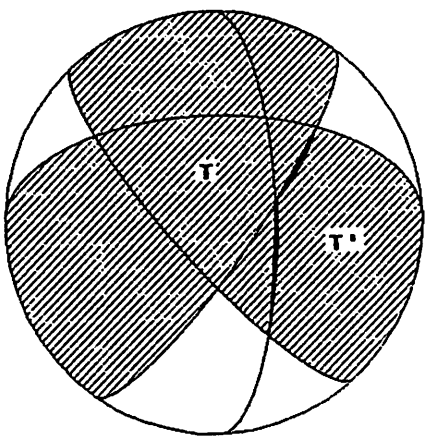


Figure 3: A lump of width 2 and the umbrella $\mathcal{U}(T', T)$

Now, shellability of $\mathcal{O}$ is an easy consequence of the following.

Lemma 6 *Let $\mathcal{L}$ a lump with $|\mathcal{L}| \geq 2$ and T, T' as in Definition 7. Then $\mathcal{L}' := \mathcal{L} \setminus T'$ is a lump of width k or $k-1$ and in the last case $\mathcal{L}'$ is complete and $k \geq 1$. Furthermore we have*

$$[\mathcal{L}'] \cap [T'] = \partial[\mathcal{L}'] \cap \partial[T'] = [\mathcal{U}(T', T)]$$

(see figure 3).

Before we are going to prove this, we first want to bring in the harvest.

Theorem 4 *Let $\mathcal{L}$ be a lump with center T of width $k < |E|$. Then $[\mathcal{L}]$ is B-shellable of dimension d.*

Proof. First note that for T' as in Definition 7 we have $T \neq -T'$ since $|sep(T', T)| < |E|$. We now proceed by induction on $|\mathcal{L}|$.

$|\mathcal{L}| = 1$ Then $[\mathcal{L}]$ is the closure of a single tope and hence B-shellable of dimension d due to Theorem 3.

$|\mathcal{L}| \geq 2$ Let T, T' as in Definition 7. Then T' is B-shellable of dimension d due to Theorem 3 and from Lemma 6 we conclude that $\mathcal{L}' := \mathcal{L} \setminus T'$ is a lump (of width $\leq k$) again and hence B-shellable of dimension d from inductive assumption. Furthermore we know that $[\mathcal{L}'] \cap [T'] = \partial[\mathcal{L}'] \cap \partial[T'] = [\mathcal{U}(T', T)]$ which is the closure of a proper umbrella of T' due to Lemma 5 and hence B-shellable of dimension $d-1$ as we have learned from Theorem 3. Alltogether this satisfies the definition of B-shellability.

◇

Corollary 1 *$\bar{\mathcal{O}}$ is B-shellable of dimension $d+1$.*

Proof. We have to prove that $\mathcal{O}$ is S-shellable of dimension d. So let T be a tope of $\mathcal{O}$ and $\mathcal{L}$ the complete lump with center T of width $|E|$. Then T' from Definition 7 equals $-T$ and $\mathcal{L}' := \mathcal{L} \setminus -T$ satisfies our assumptions from Theorem 4 and hence its closure is B-shellable of dimension d, so does $[-T]$. From Lemma 6 we conclude that $[\mathcal{L}'] \cap [-T] = \partial[\mathcal{L}'] \cap \partial[-T] = [\mathcal{U}(-T, T)]$. But we know from Lemma 5 that $\mathcal{U}(-T, T)$ is not proper and hence contains all facets of $-T$. So we get $[\mathcal{U}(-T, T)] = \partial[-T]$ which is S-shellable of dimension $d-1$ due to Theorem 3. So we are only left to prove that $\partial[\mathcal{L}'] \cap \partial[-T] = \partial[\mathcal{L}']$. The first inclusion is trivial, so let Y be a maximal element in $\partial[\mathcal{L}']$. Then Y is covered by exactly one maximal element T'' in $\mathcal{L}'$. But the interval from Y to the artificial maximum in $\bar{\mathcal{O}}$ is of dimension 1 and hence contains exactly four elements as we know from Theorem 2. But there are only three of these in $\mathcal{L}'$ hence the fourth must be $-T$ and so we have $Y \in \partial[-T]$.

◇

We still owe you the proof of Lemma 6. Remember T was the center of our lump and T' was that tope of $\mathcal{L}$ with $|sep(T', T)| = k$ we wanted to remove.

The first claim of the Lemma is trivial, we are to prove

$$[\mathcal{L}'] \cap [T'] = \partial[\mathcal{L}'] \cap \partial[T'] \tag{4.1}$$

$$\partial[\mathcal{L}'] \cap \partial[T'] = [\mathcal{U}(T', T)] \tag{4.2}$$

Proof.

ad (1) $\supseteq$ This is trivial.

$\subseteq$ Let $Y \in [\mathcal{L}'] \cap [T']$. First we claim that $|sep(Y, T)| < k$. [To see this note that $|sep(Y, T)| \leq k$ since $Y \preceq T'$. So assume $|sep(Y, T)| = k$. But then $T'' := Y \circ T = T'$ is the only tope in $\mathcal{O}$ with $Y \preceq T''$ and $|sep(T'', T)| \leq k$. Hence Y cannot be an element of $[\mathcal{L}']$ contradicting our choice of Y.]
Hence $T'' \neq T'$. Now choose a facet equator $e \in sep(T', T'')$ of T' (see Lemma 4). Then $e \in sep(T, T') \setminus supp(Y)$ and hence $Y \preceq {}_eT' \in \mathcal{L}'$. Let Z denote the common facet of T' and ${}_eT'$. From Theorem 2 we conclude that $Y \preceq Z \in \partial[\mathcal{L}'] \cap \partial[T']$ which proves (1). Furthermore note that all maximal elements in $\partial[\mathcal{L}'] \cap \partial[T']$ are facets of T'.

ad (2) $\subseteq$ Let Y be maximal in $\partial[\mathcal{L}'] \cap \partial[T']$, thus Y is a facet of T'. Let e be its facet equator. Then T' and ${}_eT'$ are the only topes covering Y (Theorem 2) hence ${}_eT' \in \mathcal{L}'$ and $e \in sep(T', T)$. This implies that $Y \in \mathcal{U}(T', T)$.

$\supseteq$ Read the former implications from right to left.

◇

Note: Günther Ziegler brought to the author's attention that shellability of Oriented Matroids was also proved by Jim Lawrence in 1984 but has not been published yet. His preprint contains the result of Lemma 6 but the proof is more involved.

References

[BaKe90] A. Bachem and W. Kern: Oriented Matroids from a Polyhedral Point of View *(in preparation)*

[BlLa78] R. G. Bland and M. Las Vergnas: Orientability of Matroids *Journal of Combinatorial Theory, Series B 24 94-123 (1978)*

[CoFu87] R. Cordovil and K. Fukuda: Oriented Matroids and Combinatorial Manifolds *Working Paper CNRS Paris (1987)*

[Dre90] A. Dress: private communication

[FoLa78] J. Folkman and J. Lawrence: Oriented Matroids *Journal of Combinatorial Theory, Series B 25 199-236 (1978)*

[Fuk82] K. Fukuda: Oriented Matroid Programming *Ph.D. Thesis University of Waterloo (1982)*

[Man82] A. Mandel: Topology of Oriented Matroids *Ph.D. Thesis University of Waterloo (1982)*

See [BaKe90] for further references.

Minimizing maximum earliness and maximum lateness on a single machine

J.A. Hoogeveen
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam
The Netherlands

A set of n jobs has to be scheduled on a single machine which can handle only one job at a time. Each job requires a given positive uninterrupted processing time p_i and has a given due date d_i. The problem is to find a schedule that minimizes a function of maximum earliness and maximum lateness that is nondecreasing in both arguments. We present $O(n^2 \log n)$ algorithms for the variant in which idle time is not allowed and for the special case in which the objective function is linear. We prove that the problem is $\mathcal{NP}$-hard if neither of these restrictions is imposed.

1980 Mathematics Subject Classification (1985 Revision): 90B35.
Key Words & Phrases: single machine scheduling, bicriteria scheduling, Pareto optimal points

1. Introduction

Suppose n independent jobs have to be scheduled on a single machine, which can handle only one job at a time. The machine is assumed to be continuously available from time 0 onwards. Job J_i $(i = 1, \ldots, n)$ requires a given positive uninterrupted processing time p_i and should ideally be completed at a given due date d_i. A *schedule* defines for each job J_i its completion time C_i such that the jobs do not overlap in their execution. Given a schedule S, the earliness and the lateness of job J_i are defined by $E_i = d_i - C_i$ and $L_i = C_i - d_i$, respectively. Correspondingly, the maximum earliness and maximum lateness are defined by $E_{\max} = \max_{1 \leqslant i \leqslant n} E_i$ and $L_{\max} = \max_{1 \leqslant i \leqslant n} L_i$, respectively. We consider the problem of finding a schedule S that minimizes the scheduling cost $F(S) = F(E_{\max}, L_{\max})$, where $F(\cdot, \cdot)$ is a given function that is nondecreasing in both arguments. We will also consider a variant of the problem in which idle time is not allowed. Extending standard notation, the first problem will be denoted by $1 \mid\mid F(E_{\max}, L_{\max})$ and the second by $1 \mid nmit \mid F(E_{\max}, L_{\max})$, where *nmit* denotes that machine idle time is forbidden.

Although the first bicriteria scheduling problem was already solved by Smith [1956], only a few bicriteria scheduling problems have been investigated since then. Most of these problems are concerned with minimizing a hierarchical type of objective function: the secondary criterion has to be minimized subject to the constraint that the primary criterion has to be minimal. Examples are minimizing the sum of completion time subject to minimal maximum lateness [Smith, 1956] and minimizing maximum lateness subject to minimum number of late jobs [Shantikumar, 1983]. Only a few of the papers on bicriteria scheduling consider simultaneous

Report BS-90xx
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

optimization, in which the criteria are transformed into a single composite objective function. An example hereof is minimizing the number of late jobs and maximum lateness simultaneously [Nelson et al., 1986]. Most contributions in the area of bicriteria scheduling concerns branch and bound algorithms. A notable exception is provided by Hoogeveen and Van de Velde [1990], who present an $O(n^2 \min\{n^2, \log \Sigma p_i\})$ time algorithm to solve $1||F(\gamma_{\max}, \Sigma C_i)$, where $\gamma_{\max} = \max\{\gamma_i \mid i = 1, \ldots, n\}$, such that all γ_i are nondecreasing in C_i. Furthermore, they present an $O(n^4)$ time algorithm to solve $1||\alpha E_{\max} + \Sigma C_i$, with $\alpha \leqslant 1$.

The organization of this paper is as follows. In Section 2 we repeat some basic theory. We also derive a dominance rule, which can be applied to both variants of the problem. In Section 3 we consider the case in which idle time is not allowed. In Section 4 we drop this constraint and analyze the general problem, which we prove to be $\mathcal{NP}$-hard in Section 5.

2. Basic concepts

The $1||F(E_{\max}, L_{\max})$ problem originates from $1||L_{\max}$ and $1|nmit|E_{\max}$, where the 'no idle time' constraint is added to $1||E_{\max}$ in order to avoid unbounded solutions. Both problems are solvable in $O(n \log n)$ time.

Earliest Due Date (EDD) Rule [Jackson, 1955]. *$L_{\max}$ is minimized by sequencing the jobs in order of nondecreasing due dates d_i.*

Minimum Slack Time (MST) Rule. *If no idle time is allowed, $E_{\max}$ is minimized by sequencing the jobs in order of nondecreasing values of $d_i - p_i$.*

The *MST* rule forms a generalization of the *EDD* rule (see Theorem 3). If both orderings coincide, then the corresponding job sequence solves both versions of $1||F(E_{\max}, L_{\max})$. However, in general both orderings will differ and it is unlikely that a single sequence minimizes both $E_{\max}$ and $L_{\max}$. Hence, in order to solve $1||F(E_{\max}, L_{\max})$ with or without idle time, we see no other way than to determine the set of feasible schedules that correspond to a Pareto optimal point with respect to the scheduling criteria $E_{\max}$ and $L_{\max}$.

Definition 1. A feasible schedule S is *Pareto optimal* with respect to the objective functions $f_1, \cdots, f_K$ if there is no feasible schedule $\bar{S}$, $\bar{S} \neq S$, such that $f_k(\bar{S}) \leqslant f_k(S)$ for all $k = 1, \ldots, K$, and $f_k(\bar{S}) < f_k(S)$ for at least one $k \in \{1, \ldots, K\}$.

Theorem 1. *Consider the composite objective function F with $F(S) = F(f_1(S), \ldots, f_K(S))$, where F is nondecreasing in all performance criteria f_k. There is a Pareto optimal schedule with respect to $f_1, \cdots, f_K$ that minimizes the function F.* □

It follows immediately from Theorem 1 that, if the number of Pareto optimal points is polynomially bounded in n and if all these points can be determined in polynomial time, then the function F can be minimized in polynomial time.

In order to find the set of Pareto optimal points, we solve the problem of minimizing $L_{\max}$ subject to $E_{\max} \leqslant E$, where E corresponds to the $E_{\max}$ value of a Pareto optimal point. This approach will be used in Section 3 and Section 4. In the remainder of this section, we present a

dominance rule that applies to both $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$ and $1 \mid E_{\max} \leqslant E \mid L_{\max}$. We need a preliminary lemma.

LEMMA 1. *Consider an arbitrary schedule S. Let J_i and J_j be two jobs, where J_i is scheduled before J_j in S. If $E_j(S) > E_i(S)$, or equivalently, $L_i(S) > L_j(S)$, then $d_j - p_j > d_i$.*

PROOF. $E_j(S) > E_i(S)$ implies $d_j - C_j(S) > d_i - C_i(S)$. As J_i is scheduled before J_j, $C_j(S) \geqslant C_i(S) + p_j$. The combination of these two inequalities yields $d_j - p_j > d_i$. □

DOMINANCE RULE. *Let J_i and J_j be two arbitrary jobs. If there exist both an MST sequence and an EDD sequence in which J_i precedes J_j, so both $d_i - p_i \leqslant d_j - p_j$ and $d_i \leqslant d_j$, where at least one of the inequalities is strict, then there exists an optimal schedule for both $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$ and $1 \mid E_{\max} \leqslant E \mid L_{\max}$ in which J_i precedes J_j.*

PROOF. The proof consists of showing that an optimal schedule S that does not satisfy the dominance rule can be transformed by applying interchanges (not necessarily adjacent) into a feasible schedule $\bar{S}$ that is optimal and that satisfies the dominance rule.

Consider an optimal schedule S which contains two jobs J_i and J_j that satisfy the conditions of the dominance rule, while J_j precedes J_i in S. Let J_i and J_j be chosen such that the jobs scheduled between J_i and J_j in S satisfy the order of the dominance rule. This implies for every job J_l, scheduled between J_j and J_i in S, that $d_j - p_j < d_l$ and $d_l - p_l < d_i$.

Consider the schedule $\bar{S}$, obtained by interchanging J_i and J_j. In order to prove that $\bar{S}$ is also an optimal feasible schedule, it suffices to prove the following two claims.

(1) $\bar{S}$ is feasible with respect to the release dates induced by the constraint $E_{\max} \leqslant E$.

(2) The lateness in $\bar{S}$ of J_j and of the jobs scheduled between J_i and J_j in S does not exceed $L_i(S) \leqslant L_{\max}(S)$.

Proof of (1). Suppose there is a job J_l in $\bar{S}$ scheduled between J_i and J_j, with $E_l(\bar{S}) > E$. As $E_i(\bar{S}) \leqslant E$, the earliness of J_l is larger than the earliness of J_i, while J_i precedes J_l. Application of Lemma 1 yields $d_l - p_l > d_i$, contradicting the assumption.

Proof of (2). Suppose there is a job J_l in $\bar{S}$ scheduled between J_i and J_j, with $L_l(\bar{S}) > L_i(S)$. As $d_i \leqslant d_j$, we have $L_i(S) = C_i(S) - d_i \geqslant C_i(\bar{S}) - d_j = C_j(\bar{S}) - d_j = L_j(\bar{S})$, so the lateness of J_l is larger than the lateness of J_j in schedule $\bar{S}$ while J_l precedes J_j. Application of Lemma 1 yields $d_j - p_j > d_l$, contradicting the assumption.

Because $d_i \leqslant d_j$ and $d_i - p_i \leqslant d_j - p_j$, the claims (1) and (2) also hold for the jobs J_i and J_j. The interchange argument can be repeated until a schedule is obtained that satisfies the dominance rule. This schedule is also feasible and optimal. □

3. PARETO OPTIMAL SCHEDULES IF NO IDLE TIME IS ALLOWED

3.1. *A polynomial algorithm for $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$*

The constraint $E_{\max} \leqslant E$ implies that $d_i - C_i \leqslant E$ for every job J_i. This is equivalent to $S_i \geqslant d_i - p_i - E$, where S_i denotes the starting time of J_i $(i = 1, \ldots, n)$. Hence, $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$ is identical to $1 \mid r_j = d_j - p_j - E, nmit \mid L_{\max}$. Although $1 \mid r_j, nmit \mid L_{\max}$ with general release dates is $\mathcal{NP}$-hard in the strong sense [Lenstra, Rinnooy

Kan and Brucker, 1977], we are able to present a polynomial algorithm for the special case that the release dates r_i are induced by the constraint $E_{\max} \leqslant E$ for an arbitrary value E of the upper bound on $E_{\max}$. The algorithm is based on the observation that, if two jobs J_i and J_j with $d_i < d_j$ can both be scheduled in the kth position of a partial schedule in which the first $k-1$ jobs have been fixed, then there exists an extension of this partial schedule that is optimal for $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$ in which J_i precedes J_j.

ALGORITHM A.

Step 0. $T := 0$; $r_j := d_j - p_j - E$ for $j = 1, \ldots, n$; $k := 1$; $U := \{J_1, \ldots, J_n\}$; $V := \varnothing$.
{Initialization: T denotes the starting time of the job in the kth position}

Step 1. For each job $J_j \in U$: if $r_j \leqslant T$ then $V := V \cup \{J_j\}$ and $U := U \setminus \{J_j\}$.
{U denotes the set of unscheduled jobs that are not allowed to start at time T, V denotes the set of unscheduled jobs that are allowed to start at time T}

Step 2. If V is empty, then stop. Otherwise, determine the job with the smallest due date in the set V. If there are ties, then choose the job with the smallest release date. If there are still ties, then choose the job with the smallest index. Suppose job J_i is chosen. This job is scheduled in the kth position.

Step 3. $T := T + p_i$; $k := k + 1$; $V := V \setminus \{J_i\}$.

Step 4. If there are unscheduled jobs left, then go to 1.

THEOREM 2. *Algorithm A yields an optimal schedule for* $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$.

PROOF. Suppose that Algorithm A yields a schedule S which is not optimal. Let $\bar{S}$ be an optimal schedule that satisfies the dominance rule.

Compare S and $\bar{S}$, starting at the front. Suppose the first difference occurs in the kth position; let J_i be scheduled in the kth position in S and let J_j be scheduled in the kth position in $\bar{S}$. Because of the construction of the schedule S, $d_i \leqslant d_j$.

Let $\hat{S}$ be the schedule that results when J_i and J_j are interchanged in $\bar{S}$. It now suffices to prove the following claims in order to prove that S is an optimal schedule that is feasible with respect to the release dates induced by the constraint $E_{\max} \leqslant E$.

(1) $\hat{S}$ is feasible with respect to the release dates induced by the constraint $E_{\max} \leqslant E$.

(2) $\hat{S}$ is also optimal.

(3) $\hat{S}$ can be transformed into a new schedule $\bar{S}$ that is optimal, feasible with respect to the release dates and equal to S with respect to the first k positions, while $\bar{S}$ satisfies the dominance rule.

Proof of (1). Analogous to the proof of claim (1) in the dominance rule.

Proof of (2). Consider an arbitrary job J_l, scheduled between J_j and J_i in $\bar{S}$. J_j precedes J_l in $\bar{S}$, so because of the dominance rule, $d_j - p_j < d_l$. Furthermore, $d_i \leqslant d_j$ because of the choice of J_i in S. Suppose $L_l(\hat{S}) > L_i(\bar{S})$. As $d_i \leqslant d_j$, $L_i(\bar{S}) \geqslant L_j(\hat{S})$. Furthermore, J_l precedes J_j in $\hat{S}$, so application of Lemma 1 yields $d_j - p_j > d_l$, contradicting the assumption that $\bar{S}$ satisfies the dominance rule. This implies for each J_l, scheduled between J_j and J_i, that the lateness of J_l in $\hat{S}$ does not exceed the value of $L_{\max}(\bar{S})$. As $L_j(\hat{S}) \leqslant L_i(\bar{S})$, this completes the proof of (2).

Proof of (3). It is possible that there exists a job J_l, scheduled between J_j and J_i in $\bar{S}$, with $d_l > d_j$ and $d_i - p_i > d_l - p_l > d_j - p_j$, implying that $\hat{S}$, obtained from $\bar{S}$ by interchanging J_j and J_i, does not satisfy the dominance rule. However, it follows immediately from the proof of the dominance rule that $\hat{S}$ can then be adjusted to a new schedule $\bar{S}$ that is also feasible and optimal, that satisfies the dominance rule, and in which the first k jobs are the same as in S.

The interchange argument can be repeated until the schedule S and the newly obtained schedule $\bar{S}$ are the same. This proves that S, obtained by the application of Algorithm A, is an optimal schedule that is feasible with respect to the release dates induced by the constraint $E_{\max} \leqslant E$, and that satisfies the dominance rule. □

THEOREM 3. *Algorithm A solves* $1 \mid L_{\max} \leqslant L$, *nmit* $\mid E_{\max}$ *to optimality.*

PROOF. Consider an arbitrary instance $V_1 = \{d_{1,1}, p_{1,1}, \ldots, d_{n,1}, p_{n,1}, L\}$ of $1 \mid L_{\max} \leqslant L$, *nmit* $\mid E_{\max}$. Now construct the following instance $V_2 = \{d_{1,2}, p_{1,2}, \ldots, d_{n,2}, p_{n,2}, E\}$ for $1 \mid E_{\max} \leqslant E$, *nmit* $\mid L_{\max}$:

$$d_{i,2} = \sum_{j=1}^{n} p_j + p_i - d_{i,1} \text{ for } i = 1, \ldots, n,$$

$$p_{i,2} = p_{i,1} \text{ for } i = 1, \ldots, n,$$

$$E = L.$$

Suppose Algorithm A produces a schedule S as an optimal schedule for V_2. An optimal schedule $\bar{S}$ for V_1 is then obtained by reversing S, since

$$E_i(\bar{S}) = d_{i,1} - C_i(\bar{S}) = \sum_{j=1}^{n} p_j + p_i - d_{i,2} - C_i(\bar{S}) = C_i(S) - d_{i,2} = L_i(S).$$

This implies that $L_{\max}(S) \leqslant L$ if and only if $E_{\max}(\bar{S}) \leqslant E$, and S is optimal and feasible if and only if S is optimal and feasible. □

3.2. *Determination of Pareto optimal points with respect to* $E_{\max}$ *and* $L_{\max}$

We now present an algorithm to determine all values E of $E_{\max}$ that may correspond to a Pareto optimal point. Once such a value E is known, the corresponding value of $L_{\max}$ can be determined in $O(n \log n)$ time by solving the corresponding $1 \mid E_{\max} \leqslant E$, *nmit* $\mid L_{\max}$ problem by Algorithm A. Furthermore we prove that the number of Pareto optimal points with respect to $E_{\max}$ and $L_{\max}$ is no more than n, and that at most n values E of $E_{\max}$ have to be considered to determine all Pareto optimal points. We first prove the following lemma.

LEMMA 2. *Consider an arbitrary job* J_k *in* S, *where* S *is the schedule constructed by Algorithm A to solve* $1 \mid E_{\max} \leqslant E$, *nmit* $\mid L_{\max}$. *There are no two jobs* J_i *and* J_j, *scheduled before* J_k, *with a due date larger than* d_k.

PROOF. Suppose the opposite and assume that there are two such jobs J_i and J_j. Without loss of generality, let J_i be scheduled before J_j. Because of the construction of the schedule, job J_k may not be chosen by Algorithm A at time $C_i(S)$, so $d_k - p_k - E > C_i(S) \geqslant d_i - E$, implying $d_k - p_k > d_i$, so $d_k > d_i$, contradicting the assumption. □

Given an optimal schedule S for $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$, obtained by Algorithm A, where E is an arbitrary value of the upper bound on $E_{\max}$, it is possible to decrease $L_{\max}$ only by interchanging two jobs that are not scheduled in *EDD* order. We prove that S can be partitioned into blocks that have the property that an interchange, necessary to decrease $L_{\max}$, can only take place within such a block. Partition the schedule S into blocks according to the following algorithm.

PARTITIONING ALGORITHM.

Step 0. Start at the beginning of the schedule.

Step 1. Let J_i be the first job in a block. Compare the due date of job J_i with the due date of its successors, until a job is found that does not have a smaller due date. Let this be J_k. The block contains job J_i and all its successors up to J_k.

Step 2. Proceed until the schedule has been completely partitioned into blocks.

PROPOSITION 1. *Let S be an optimal schedule for the $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$ problem obtained by Algorithm A, where E is an arbitrary upper bound on $E_{\max}$, with $E \geqslant E^{MST}$, where E^{MST} is equal to $E_{\max}(MST)$. Partition S into blocks, according to the Partitioning algorithm. Any block B has the following properties.*

(1) *If job J_i is the first job in B, then all jobs J_j in S with smaller due date, scheduled after J_i, also belong to block B.*

(2) *The jobs in B are scheduled in the following order: the job with the largest due date is scheduled first, the other jobs are scheduled in EDD order.*

PROOF. (1) Suppose that there exists a job J_j with $d_j < d_i$, scheduled after J_i, that does not belong to B. According to the Partitioning algorithm, there must exist a job J_l, scheduled between J_i and J_j, with $d_l > d_i > d_j$. But then, both J_i and J_l have larger due date and are scheduled before J_j, contradicting Lemma 2. This contradiction proves Property 1.

(2) Property 2 follows immediately from Lemma 2 and the Partitioning algorithm. □

THEOREM 4. *Let E_1 and E_2 be two arbitrary values of the upper bound on $E_{\max}$, with $E_1 \leqslant E_2$. Let S_1 and S_2 be the optimal schedules obtained by applying Algorithm A to $1 \mid E_{\max} \leqslant E_1, nmit \mid L_{\max}$ and $1 \mid E_{\max} \leqslant E_2, nmit \mid L_{\max}$, respectively. Partition S_1 into blocks according to the Partitioning algorithm. Let B be an arbitrary block of S_1, let T_1 and T_2 be the starting and completion time of block B in S_1, respectively. Then the jobs belonging to B are processed in S_2 during the interval $[T_1, T_2]$.*

PROOF. Consider the first block B of the schedule S_1. Let J_j be an arbitrary job that does not belong to B. As $E_2 \geqslant E_1$, it is feasible to schedule all jobs of B in the same position as in S_1. As J_j does not belong to B, its due date is at least as large as the due date of the first job in B and therefore Algorithm A will not choose J_j until all jobs that belong to B have been scheduled. This argument can be repeated for the second block in S_1 and so on, until only the last block remains. □

THEOREM 5. *Let S be an optimal schedule for $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$ obtained by Algorithm A, where E is an arbitrary upper bound on $E_{\max}$. Let B be a block that contains a job J_i with $L_i = L_{\max}(S)$. In order to decrease $L_{\max}(S)$, it is necessary to increase E such that another job within block B can be scheduled first.*

PROOF. $L_{\max}(S)$ can only be decreased by decreasing the completion time of job J_i. Application of Theorem 4 implies that a decrease of the completion time of job J_i has to be achieved by changing its position within the block B. As all jobs in B, except the job in the first position, are scheduled in *EDD* order, another job has to be scheduled in the first position of B in order to change the scheduling order in B. □

Theorem 4 and 5 provide the basis for the algorithm we present for determining all Pareto optimal points. First we prove that the number of Pareto optimal points is no more than n.

THEOREM 6. *Let S_1 and S_2 be two schedules obtained by Algorithm A, which both correspond to a Pareto optimal point, (E_1, L_1) and (E_2, L_2), respectively. Suppose $E_1 < E_2$. Partition S_1 and S_2 into blocks by applying the Partitioning algorithm. The number of blocks in which S_1 has been partitioned is smaller than the number of blocks in which S_2 has been partitioned.*

PROOF. Application of Theorem 4 yields that S_2 has been partitioned in at least as many blocks as S_1. Furthermore, Theorem 5 implies that, in order to achieve a lower value of $L_{\max}$, at least one of the blocks in which S_1 has been partitioned, must have been split in at least two blocks in S_2. □

COROLLARY 6.1. *The number of Pareto optimal points is bounded by n, and this bound is tight.*

PROOF. The number of blocks is at most equal to the number of jobs. From Theorem 6 it follows immediately that there are no two different Pareto optimal points, such that the corresponding schedules, obtained by Algorithm A, are partitioned by the Partitioning algorithm, into the same number of blocks. This implies that the number of Pareto optimal points is bounded by n.

The following example shows that the bound is tight:

$$p_i = 1 \quad \text{for } i = 1, \ldots, n-1,$$

$$d_1 = 2,\ d_{i+1} = d_i + i + 2 \text{ for } i = 1, \ldots, n-2,\ d_n = p_n = d_{n-1} + 1. \quad \square$$

From Theorem 5 it follows immediately that a new Pareto optimal point can only be obtained by increasing the value E of the upper bound on $E_{\max}$ such that for every block B that contains a job J_i with $L_i = L_{\max}$ another job can be scheduled in the first position in B. This observation forms the basis for an algorithm that, given an optimal schedule S for $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$, determines the next $E_{\max}$ value that corresponds to a Pareto optimal point.

ALGORITHM NEXT E

Step 0. Partition S into blocks, according to the Partitioning algorithm.

Step 1. Determine for each block B that contains a job J_i with $L_i(S) = L_{\max}(S)$ the new value of E that is necessary to schedule another job in B in the first position. If J_i is the only job in B, then $L_{\max}$ cannot be decreased. Stop.

Step 2. Choose the maximum of the new values found at Step 1. This maximum is the new value E.

The combination of all properties derived above leads to Algorithm B that determines all Pareto optimal points in $O(n^2 \log n)$ time. As the proof of this algorithm should be trivial by now, we omit it.

ALGORITHM B

Step 0. Solve $1 \mid\mid L_{\max}$; this yields L^{EDD} and solve $1 \mid nmit \mid E_{\max}$; this yields E^{MST}.

Step 1. Solve $1 \mid E_{\max} \leqslant E^{MST}, nmit \mid L_{\max}$ by applying Algorithm A; this leads to the first Pareto optimal point.

Step 2. If L^{EDD} has been reached, then go to 4. Otherwise, determine the next value E of the upper bound on $E_{\max}$ that corresponds to a Pareto optimal point by applying Algorithm Next E.

Step 3. Solve $1 \mid E_{\max} \leqslant E, nmit \mid L_{\max}$; this leads to a new Pareto optimal point, go to 2.

Step 4. All Pareto optimal points have been determined. The $1 \mid nmit \mid F(E_{\max}, L_{\max})$ problem can be solved by computing the value of $F(E_{\max}, L_{\max})$ for all of these points.

4. PARETO OPTIMAL SCHEDULES IF IDLE TIME IS ALLOWED

We prove that the $1 \mid E_{\max} \leqslant E \mid L_{\max}$ problem for a given value E can be solved in $O(n^2 \log n)$ time. We show that the trade-off curve of $E_{\max}$ and $L_{\max}$, defined as the curve that connects all points (E,L) where L is the outcome of $1 \mid E_{\max} \leqslant E \mid L_{\max}$, is piecewise linear with gradient alternately -1 and 0. As every point on the trade-off curve with negative gradient is not dominated, the number of Pareto optimal points with respect to $E_{\max}$ and $L_{\max}$ is no longer polynomially bounded by n. Hence, we do not know how to solve $1 \mid\mid F(E_{\max}, L_{\max})$ in polynomial time, unless we add the constraint that $F(E_{\max}, L_{\max})$ is linear.

Consider the $1 \mid E_{\max} \leqslant E \mid L_{\max}$ problem. As $L_{\max}$ is a regular performance measure, we may restrict our attention to *active* schedules. An active schedule is a schedule in which no job can start earlier without increasing the completion time of at least one other job.

The possibility of inserting idle time in a schedule has an important consequence. Consider a partial schedule without idle time in which the first $k-1$ jobs have been fixed. Instead of scheduling the job selected by Algorithm A in the kth position, it now may be advantageous to wait until another job with a smaller due date becomes available. So, if we stick to a schedule without idle time, it may be advantageous to increase the upper bound E on the earliness of the job that is to be scheduled in the kth position. The increase of the upper bound E should be considered for every position separately. Let $J_{[i]}$ denote the job in the ith position in the schedule and let K_i denote the upper bound on the earliness of job $J_{[i]}$. We analyze the $1 \mid E_{[i]} \leqslant K_i, nmit \mid L_{\max}$ problem instead of the $1 \mid E_{\max} \leqslant E \mid L_{\max}$ problem. As the sequence

without idle time has to be made feasible with respect to $E_{\max} \leqslant E$ by inserting idle time, there is no use in considering vectors $K = (K_1, \ldots, K_n)$ that do not satisfy $K_i \leqslant K_{i+1}$ for $i = 1, \ldots, n-1$.

Let K be a vector of upper bounds. Let S be the corresponding sequence, S is obtained by solving $1 \mid E_{[i]} \leqslant K_i, \textit{nmit} \mid L_{\max}$. Let $S(E)$ be the corresponding active schedule that is feasible with respect to the constraint $E_{\max} \leqslant E$. We will solve $1 \mid E_{\max} \leqslant E \mid L_{\max}$ for every value E by determining the set of vectors $\{K^1, \ldots, K^m\}$ such that for every value E one of the corresponding schedules $S^1(E), \ldots, S^m(E)$ solves $1 \mid E_{\max} \leqslant E \mid L_{\max}$. The vectors $K^1, \ldots, K^m$ have to be minimal, so there is no vector $\bar{K}$ smaller than K^j such that the schedule S^j is feasible with respect to $E_{[i]} \leqslant \bar{K}_i$, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Therefore, $K_j^k = \max\{E_{[i]}(S^j) \mid i = 1, \ldots, k\}$ for $j = 1, \ldots, m$ and $k = 1, \ldots, n$.

The $1 \mid E_{[i]} \leqslant K^i, \textit{nmit} \mid L_{\max}$ problem is a straightforward generalization of the $1 \mid E_{\max} \leqslant E, \textit{nmit} \mid L_{\max}$ problem. Therefore, an optimal sequence for the $1 \mid E_{[i]} \leqslant K_i, \textit{nmit} \mid L_{\max}$ problem has almost the same properties and can be determined in the same fashion as an optimal sequence for $1 \mid E_{\max} \leqslant E, \textit{nmit} \mid L_{\max}$.

LEMMA 3. *Given a vector $K = (K_1, \ldots, K_n)$ with $K_i \leqslant K_{i+1}$ for $i = 1, \ldots, n-1$, there exists an optimal schedule S for $1 \mid E_{[i]} \leqslant K_i, \textit{nmit} \mid L_{\max}$ that satisfies the dominance rule.*

PROOF. As $K_i \leqslant K_{i+1}$ for $i = 1, \ldots, n-1$, the proof of Lemma 3 is analogous to the proof of the dominance rule. □

We now adjust Algorithm A such that it can be applied to solve $1 \mid E_{[i]} \leqslant K_i, \textit{nmit} \mid L_{\max}$.

ALGORITHM AA.

Step 0. $T := 0$; $k := 1$; $U := \{J_1, \ldots, J_n\}$; $V := \varnothing$.

{Initialization: T denotes the starting time of the job in the kth position}

Step 1. For each job $J_j \in U$: if $d_j - p_j - K_k \leqslant T$ then $V := V \cup \{J_j\}$ and $U := U \setminus \{J_j\}$.

{U denotes the set of unscheduled jobs that are not allowed to start at time T, V denotes the set of unscheduled jobs that are allowed to start at time T}

Step 2. If V is empty, then stop. Otherwise, determine the job with the smallest due date in the set V. If there are ties, then choose the job with the smallest value of $d_j - p_j$. If there are still ties, then choose the job with the smallest index. Suppose job J_i is chosen. This job is scheduled in the kth position.

Step 3. $T := T + p_i$; $k := k + 1$; $V := V \setminus \{J_i\}$.

Step 4. If there are unscheduled jobs left, then go to 1.

THEOREM 7. *Algorithm AA yields an optimal schedule for $1 \mid E_{[i]} \leqslant K_i, \textit{nmit} \mid L_{\max}$.*

PROOF. The proof of Theorem 7 is analogous to the proof of Theorem 2. □

We now come to the point of how to determine the set of upper bound vectors $\{K^1, \ldots, K^m\}$. We will follow the same approach as in Section 3.2.

Consider a vector K from the set $\{K^1, \ldots, K^m\}$. Let k and l be such that $K_{k-1} < K = \cdots = K_l < K_{l+1}$. The set of positions $\{j, \ldots, k\}$ is called a *group of positions*, the set of jobs $\{J_{[k]}, \ldots, J_{[l]}\}$ is called a *group* of jobs. Note that the corresponding schedule $S(E)$ contains no idle time within a group of jobs. Define the maximum lateness within the group G as $L(G)_{\max} = \max\{L_{[i]}(S) \mid J_{[i]} \in G\}$ and define $K(G)$ as the K-value of a position within G.

PROPOSITION 2. *Let S^j be the sequence obtained by applying Algorithm AA to $1 \mid E_{[i]} \leqslant K_i$, nmit $\mid L_{\max}$, where $K^j \in \{K^1, \ldots, K^m\}$. Partition S_j in groups, let G_1 and G_2 be two arbitrary groups of jobs, where G_1 is completed before G_2. Let $J_{[i]}$ and $J_{[k]}$ be two arbitrary jobs in G_1 and G_2, respectively. Then $d_{[i]} < d_{[k]}$.*

PROOF. Let $J_{[l]}$ be the first job in G_2; therefore $J_{[l]}$ must start at time $d_{[l]} - p_{[l]} - K^j_{[l]}$. The starting time of $J_{[l]}$ is larger than or equal to the completion time of $J_{[i]}$, which is at least equal to $d_{[i]} - K^j_{[i]} > d_{[i]} - K^j_l$. Therefore, $d_{[l]} - p_{[l]} > d_{[i]}$, and as S^j satisfies the dominance rule stated in Lemma 3, $d_{[j]} > d_{[l]} - p_{[l]} > d_{[i]}$. □

Proposition 2 enables us to state Theorem 8, the analogue of Theorem 4.

THEOREM 8. *Let K^j and K^k be two arbitrary vectors from the set $\{K^1, \ldots, K^m\}$. Let K^k be the larger of the two. Let S^j and S^k be two optimal sequences obtained by applying Algorithm AA to $1 \mid E_{[i]} \leqslant K^j_i$, nmit $\mid L_{\max}$ and $1 \mid E_{[i]} \leqslant K^k_i$, nmit $\mid L_{\max}$, respectively. Partition S^j and S^k in groups. Let G_1 and G_2 be two arbitrary groups of jobs in S^j, where G_1 is completed before G_2 in S^j. Let $J_{[a]}$ and $J_{[b]}$ be two arbitrary jobs in G_1 and G_2, respectively. Then $J_{[a]}$ precedes $J_{[b]}$ in S^k.*

PROOF. The proof follows from Proposition 2 and the way the jobs are chosen in Algorithm AA. □

From Theorem 8 it follows immediately that, if we start with a vector K for the upper bound on $E_{[i]}$, then the only way to decrease $L(G)_{\max}$ is to increase the value of the upper bound for the whole group G or for a part of the group. This observation provides the basis for the following algorithm.

ALGORITHM NEXT K.

Step 0. Let K be a given vector of upper bounds on $E_{[i]}$. Let S be the sequence obtained by applying Algorithm AA to $1 \mid E_{[i]} \leqslant K_i$, nmit $\mid L_{\max}$.

Step 1. Let G be the first group in the schedule that attains $\max\{L(G)_{\max} + K(G)\}$. Partition this group of jobs into blocks, by applying the Partitioning algorithm.

Step 2. Determine the set of blocks $\mathfrak{B}$ in G that contain a job J_i with $L_i(S) = L(G)_{\max}$.

Step 3. Determine for each block B in $\mathfrak{B}$ how much the upper bound $K(G)$ has to be increased in order to let another job within B be scheduled in the first position in B. Let this value be equal to $K(B)$. If B consists of a single job, then $K(B) = \infty$ and hence, $L(G)_{\max}$ cannot be decreased. Stop.

Step 4. The next vector of upper bounds K can be computed from the old upper bound vector as follows. Let the first block in $\mathfrak{B}$ start in the $(k+1)$th position. The first k elements stay the same. Now consider the remaining positions in G, suppose this are the positions $k+1, \ldots, l$. The new upper bound value K_{i+1} becomes equal to K_i, unless a block $B \in \mathfrak{B}$, with $K(B) > K_i$ starts at position $i+1$. The elements of the new upper bound vector

K, corresponding to positions after G become equal to the maximum of K_l and their old value.

Because every group of jobs G has the same K-value for every job J_i in G, the correctness of this algorithm follows from Theorem 5.

Note that K_i^j cannot be smaller than K_i^{j+1} $(i = 1, \ldots, n; j = 1, \ldots, m-1)$, otherwise $L_{\max}(S^{j+1})$ will be larger than $L_{\max}(S^j)$.

We are now able to formulate an algorithm to determine all vectors $K^1, \ldots, K^m$ and the corresponding sequences $S^1, \ldots, S^m$. Define the vector K^{MST} such that $K_i^{MST} = \max\{E_{[j]}(MST) \mid j = 1, \ldots, i\}$ for $i = 1, \ldots, n$.

Algorithm BB.

Step 0. Determine the vector K^{MST}; $l := 1$; $K^l := K^{MST}$

Step 1. Solve $1 \mid E_{[i]} \leqslant K_i^l, nmit \mid L_{\max}$ by applying Algorithm AA, this yields sequence S^l.

Step 2. $l := l + 1$. Compute the next vector K^l by applying Algorithm Next K. If $K < \infty$, then go to 1.

Step 3. All vectors $K \in \{K^1, \ldots, K^m\}$ have been determined.

The set of vectors $\{K^1, \ldots, K^m\}$ can be screened to remove all vectors that lead to dominated sequences. A sequence S^{j+1} is dominated by sequence S^j if $L_{\max}(S^j) \leqslant L_{\max}(S^{j+1})$.

THEOREM 9. *Let K^j and K^k be two arbitrary vectors from the set $\{K^1, \ldots, K^m\}$. Let K^k be the larger of the two. Let S^j and S^k be the optimal sequences obtained by applying Algorithm AA to $1 \mid E_{[i]} \leqslant K_i^j, nmit \mid L_{\max}$ and $1 \mid E_{[i]} \leqslant K_i^k, nmit \mid L_{\max}$, respectively. Partition S^j and S^k into blocks according to the Partitioning algorithm. Then S^k is partitioned into more blocks than S^j so, as the number of blocks is bounded by n, Algorithm BB computes at most n schedules.*

PROOF. The proof is analogous to the proof of Theorem 6 and Corollary 6.1. □

Let $S^j(E)$ be the active schedule obtained by inserting idle time in S^j to make $S^j(E)$ feasible with respect to the constraint $E_{\max} \leqslant E$, for $j = 1, \ldots, m$. Let $J_{[i]}$ be an arbitrary job in $S^j(E)$. We now have that $L_{[i]}(S^j(E)) = L_{[i]}(S^j) + \max\{0, K_i^j - E\}$, so the trade-off curve of $L_{\max}(S^j(E))$ and E forms a continous, piecewise linear step-function with gradient alternately -1 and 0, for $j = 1, \ldots, m$. Furthermore, the number of breakpoints in every trade-off curve is no more than n. These trade-off curves can be combined to one trade-off curve by choosing for each value of E the minimum value of $L_{\max}(S^j(E))$ for $j = 1, \ldots, m$. The number of breakpoints in this trade-off curve is no more than $mn \leqslant n^2$. As $1 \mid E_{\max} \leqslant E \mid L_{\max}$ is solved by one of the schedules $S^1(E), \ldots, S^m(E)$ for every value of E, the trade-off curve derived above is exactly equal to the trade-off curve of $L_{\max}$ and E, where $L_{\max}$ is equal to the outcome of the $1 \mid E_{\max} \leqslant E \mid L_{\max}$ problem.

5. Solving the $1||F(E_{\max}, L_{\max})$ problem is $\mathcal{NP}$-hard

In this section we prove that $1||F(E_{\max}, L_{\max})$ is $\mathcal{NP}$-hard in the strong sense. First, we need to prove that the problem of minimizing an arbitrary function $f(x)$, where x belongs to an arbitrary set P of integers, is $\mathcal{NP}$-hard, by showing that the corresponding decision problem is $\mathcal{NP}$-complete. The reduction is from the Hamiltonian circuit problem [Schrijver, 1989].

Hamiltonian circuit problem [Garey and Johnson, 1979]: Given a graph $G = (V,H)$, does G contain a Hamiltonian circuit?

We start by describing a reduction from the Hamiltonian circuit problem to the problem of minimizing an arbitrary function $f(x)$, where x belongs to an arbitrary set P of integers. Let $G = (V,H)$ be an arbitrary graph and let the edges in H be numbered $1, \ldots, |H|$, where $|H|$ denotes the cardinality of H. Define $P = \{0, \ldots, 2^{|H|}\}$. Every integer $x \in P$ can be described by $|H|$ zeros and ones, by using binary encoding. Further, define for every $x \in P$ a subset $P_x \subset H$ in the following way: the ith edge in H belongs to P_x if and only if the ith digit in the binary representation of x is equal to 1. Now we define the following function $f(x)$.

$$f(x) = \begin{cases} 0 & \text{if } |P_x| = n \text{ and if } P_x \text{ is Hamiltonian,} \\ 1 & \text{otherwise.} \end{cases}$$

Clearly, the value of $f(x)$ can be established in every point x in polynomial time.

Theorem 10. *Given a set of integers S and a nonnegative integer y, the problem of deciding whether there exists an integer $x \in S$ with $f(x) \leqslant y$ is $\mathcal{NP}$-complete.*

Proof. The decision problem is clearly in $\mathcal{NP}$. For any given instance of the Hamiltonian circuit problem, we construct a set of integers S and a function $f(x)$ as described above, and set $y = 0$. This reduction requires polynomial time. The decision problem will be answered affirmatively if and only if the graph G contains a Hamiltonian circuit. □

Theorem 11. *The $1||F(E_{\max}, L_{\max})$ problem is $\mathcal{NP}$-hard in the strong sense.*

Proof. The trade-off curve is piecewise linear with gradient -1 and 0 alternately. This implies that the number of Pareto optimal points with respect to the criteria $E_{\max}$ and $L_{\max}$ is unbounded, as every value $E \in \mathbb{Z}$, with E not larger than the E-value of the first breakpoint, corresponds to a Pareto optimal point. Therefore, we are able to select $2^{|H|}$ consecutive Pareto optimal points (with H equal to the edge set of an arbitrary graph) and therefore, we can carry out the reduction from the Hamiltonian Circuit problem, with $F(E_{\max}, L_{\max}) = f(E - c)$, where c is such that $0 \leqslant E - c < 2^{|H|}$ for every selected E-value. □

Note that if we impose the restriction that $E_{\max}$ has to be nonnegative, then only a pseudo-polynomial number of Pareto optimal points remains, so solving $1||F(E_{\max}, L_{\max})$ takes at most pseudo-polynomial time. Suppose that a polynomial algorithm exists for this restricted problem. In that case, given a graph $G = (V,H)$, all processing times can be multiplied by a factor $O(2^{|H|})$, after which we can select $2^{|H|}$ consecutive Pareto optimal points (the idle time

can still be changed by one unit at a time) and we can carry out the reduction as described above. Therefore, even in case $E_{\max}$ is bounded from below there is no polynomial algorithm for $1 \mid\mid F(E_{\max}, L_{\max})$, unless $\mathcal{P}=\mathcal{NP}$. Further, note that the reduction from the Hamiltonian circuit problem is not polynomial anymore, when $E_{\max}$ is assumed to be nonnegative and therefore, we cannot conclude that this special case of $1 \mid\mid F(E_{\max}, L_{\max})$ is $\mathcal{NP}$-hard in the strong sense.

However, if we consider linear objective functions $F(E_{\max}, L_{\max}) = \alpha_1 E_{\max} + \alpha_2 L_{\max}$, with $\alpha_1, \alpha_2 \geqslant 0$, then the situation is much brighter. If $\alpha_1 > \alpha_2$, then the optimum cost value will be equal to $-\infty$, otherwise an optimum is found in one of the breakpoints. As every breakpoint can be found in constant time, $1 \mid\mid \alpha_1 E_{\max} + \alpha_2 L_{\max}$ can be solved in $O(n^2 \log n)$ time, the time needed to determine the schedules $S^1, \ldots, S^m$.

ACKNOWLEDGEMENT

The author would like to thank Jan Karel Lenstra for his helpful comments.

REFERENCES

M.R. GAREY, D.S. JOHNSON (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

J.A. HOOGEVEEN, S.L. VAN DE VELDE (1990). *Some results on single machine multi criteria scheduling*, Working Paper, Centre for Mathematics and Computer Science, Amsterdam.

J.R. JACKSON (1955). *Scheduling a Production Line to Minimize Maximum Tardiness*, Research Report 43, Management Science Research Project, University of California, Los Angeles.

J.K. LENSTRA, A.H.G. RINNOOY KAN, P. BRUCKER (1977). Complexity of machine scheduling problems. *Ann. Discrete Math. 1*, 343-362.

R.T. NELSON, R.K. SARIN, R.L. DANIELS (1986). Scheduling with multiple performance measures: the one-machine case. *Management Science 32*, 464-479.

A. SCHRIJVER (1989). Private communication.

J.G. SHANTHIKUMAR (1983). Scheduling n jobs on one machine to minimize the maximum tardiness with minimum number tardy. *Computers and Operations Research 10*, 255-266.

W.E. SMITH (1956). Various Optimizers for single-stage production. *Naval Research Logistics Quarterly 1*, 59-66.

$O(m \cdot n)$ ISOMORPHISM ALGORITHMS FOR CIRCULAR–ARC GRAPHS AND CIRCLE GRAPHS

Wen–Lian Hsu †

Department of Industrial Engineering and Management Sciences, Northwestern University and Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC

Abstract

The circular endpoint sequence of arcs in a model for a circular–arc graph is usually far from unique. We present a notion of "normalized models" to eliminate trivial endpoint variations and characterize those circular–arc graphs which have unique normalized models. Based on this, an $O(m \cdot n)$ algorithm is given for the recognition and isomorphism problems on circular–arc graphs. Our techniques involve a transformation from a given circular–arc graphs to a circle graph and two decomposition operations: the join and the substitution. This approach has the following advantages: (1) it is conceptually much simpler than Tucker's $O(n^3)$ recognition algorithm; (2) it exploits the similarity between circle graphs and circular–arc graphs in a natural fashion; (3) it yields a simple isomorphism algorithm. As an easy consequence, our approach also yields an $O(m \cdot n)$ isomorphism algorithm for circle graphs.

1. Introduction

Intersection graphs have recently received considerable attention in the study of algorithmic graph theory and their applications (see, e.g. [1,8,9,12,16,22]). Well–known special classes of intersection graphs include interval graphs, chordal graphs, circular–arc graphs, permutation graphs, circle graphs and etc. The isomorphism problem is polynomially solvable on interval graphs, and isomorphism–complete on chordal graphs. In this paper, we present $O(m \cdot n)$ isomorphism algorithms for circle graphs and circular–arc graphs. Our approach is closely related to their underlying representation trees.

Tucker [8] has given an $O(n^3)$ algorithm for recognizing circular–arc graphs. However, the algorithm and its proof are very involved and it does not render an isomorphism algorithm. Spinrad [18] gave a simpler recognition algorithm for those circular–arc graphs which can be covered by two cliques. We present an $O(m \cdot n)$ recognition algorithm which naturally yields an isomorphism algorithm for circular–arc graphs. A special case of our algorithm solves the isomorphism problem for circle graphs. A key part of our algorithm is to reduce the circular–arc graph recognition problem to the circle graph recognition problem.

The difficulty in dealing with circular–arc graphs lies largely in that even a simple circular–arc graph (or interval graphs) can have an exponential number of arc representations. We propose a notion of ***normalized representations*** with the property that the relative endpoint positions of any two arcs in such representations are prescribed by the adjacency relationships of their corresponding vertices in G. By restricting to these special representations, we can eliminate trivial endpoint variations. Through graph decomposition, we provide a characterization of those circular–arc graphs which possess "unique" normalized representations. Furthermore, the decomposition yields unique tree representations that are used in the isomorphism testing. We believe our approach has the following advantages: (1) it is conceptually very simple; (2) it exploits the similarity

between circle graphs and circular–arc graphs in a very natural fashion; (3) it yields an simple isomorphism algorithm for circular–arc graphs.

Denote a graph G by a pair (V,E), where V (or V(G)) denotes the finite vertex set of G and E (or E(G)) denotes a set of edges connecting vertices of G. Let n = |V| and m = |E|. Let F be a family of nonempty sets. The ***intersection graph*** of F is obtained by representing each set in F by a vertex and connecting two vertices by an edge if and only if their corresponding sets have a nonempty intersection. The intersection graphs of a family of intervals (resp. arcs, chords) is called an ***interval graph*** (resp. circular–arc graph, circle graphs). Each of these representations using intervals, arcs or chords is called a ***model*** for the corresponding graph. Two models for a circular–arc graph (resp. interval) are said to be ***equivalent*** if a circular (resp. left–right) label sequence of one model can be obtained from that of the other through rotation or reflection (resp. through reflection). A circular–arc (resp. interval) graph G is said to have a ***unique*** circular–arc (resp. interval) model if all circular–arc (resp. interval) models for G are equivalent. It is easy to see that a given circular–arc graph can have an exponential number of non–equivalent models.

The main technique used in our recognition algorithm is graph decomposition. This is a useful divide–and–conquer scheme for attacking many optimization and recognition problems. Gabor, Hsu and Supowit [5] and independently, Bouchet [2] used the join decomposition to recognize circle graphs. We shall adopt both the substitution and the join decompositions in recognizing circular–arc graphs.

In the next section we introduce the notion of normalized models for a circular–arc graph. The two decomposition schemes are discussed in Section 3. Section 4 is devoted to the transformation of circular–arc graphs to circle graphs. The decomposition of circular–arc graphs are discussed in Sections 5 and 6. Finally, based on those decomposition schemes, we present an isomorphism algorithm for circular–arc graphs in Section 7.

2. Normalized models for circular–arc graphs

In this section, we show that there exist circular–arc models in which the overlapping relationships (described below) of any two arcs are strictly dictated by their corresponding neighborhood structures in the graph.

Definition. *Given a circular–arc model D, there are four possible overlapping relationships on a pair of arcs $u_1 = (a_1,b_1)$ and $u_2 = (a_2,b_2)$ in D defined as follows:*

(1) *u_1 is said to be **independent of** u_2 if they do not overlap.*

(2) *u_1 is said to be **contained in** u_2 (respectively, contains) if segment (a_2,b_2) contains (respectively, is contained in) segment (a_1,b_1).*

(3) *u_1 and u_2 are said to **cross** each other if their four endpoints appear alternately on the circle.*

(4) *u_1 and u_2 are said to **cover the circle** if the two segments (a_1,b_2) and (a_2,b_2) together cover the whole circle.*

Denote by ***N(v)*** the set consisting of v and all vertices adjacent to v in G. The related vertex adjacency relationships can be defined as follows.

Definition. *Let v_1 (resp. v_2) denote the vertex in a circular–arc graph G corresponding to arc u_1 (resp. u_2). Two vertices v_1 and v_2 in a graph G are said to be*

(i) ***strictly adjacent*** *if v_1 is adjacent to v_2 but neither $N(v_1)$ nor $N(v_2)$ is contained in the other.*

(ii) ***strongly adjacent*** *if v_1 and v_2 are strictly adjacent and every w in $V(G)\backslash N(v_1)$ satisfies that $N(w) \subseteq N(v_2)$ and every w' in $V(G)\backslash N(v_2)$ satisfies that $N(w') \subseteq N(v_1)$.*

(iii) ***similar*** *if $N(v_1)\backslash\{v_1\} = N(v_2)\backslash\{v_2\}$.*

These relationships can be determined by finding the partial order (based on the "⊇"

relation) on the neighborhood sets N(v)'s for all v, which takes at most $O(m \cdot n)$ time. A vertex v in G is said to be an ***A-vertex*** if N(v) = V(G).

Definition. *Let G be a circular-arc graph containing neither similar pairs nor A-vertices. A circular-arc model D for G is said to be a* ***normalized model (N-model)*** *if every pair of arcs u_1 and u_2 in D satisfy the following conditions:*

u_1 is independent of u_2	$\Leftrightarrow$	*v_1 is not adjacent to v_2*
u_1 is contained in u_2	$\Leftrightarrow$	*$N(v_1) \subseteq N(v_2)$*
u_1 and u_2 cover the circle	$\Leftrightarrow$	*v_1 and v_2 are strongly adjacent*
u_1 crosses u_2	$\Leftrightarrow$	*v_1 and v_2 are strictly but not strongly adjacent*

Similarly, one can define a normalized interval model for interval graphs. There is an $O(m \cdot n)$ algorithm that transforms any circular-arc model of G into an N-model.

Theorem 2.1. *Let G be a circular-arc graph containing neither similar pairs nor A-vertices. Then there exists an N-model for G. If, in particular, G is an interval graph, then there also exists an interval N-model for G.*

From now on, we consider only normalized models for circular-arc graphs. To characterize graphs having unique N-models, we need to apply certain graph decompositions, which is discussed in the next section.

3. The substitution decomposition and the join decomposition

Substitution decomposition, also known as ***modular decomposition***, has been discussed in many papers (see, e.g. [8,17,19]). In [19], an $O(n^2)$ algorithm is given for decomposing a graph completely into inseparable components. We shall follow closely the notations of Spinrad [17,19].

Definition. *A* ***module*** *is a subset M of V(G) such that* each vertex u in $V \backslash M$ is either adjacent to all vertices in M or adjacent to no vertex in M. *M is a* ***connected*** *module iff G[M] is connected. M is* ***complement-connected*** *iff the complement of G[M] is connected. An unconnected module is called a* ***parallel*** *module. A connected module whose complement is unconnected is called a* ***series*** *module. A module which is both connected and complement-connected is called a* ***neighborhood*** *module.*

Definition. *A graph G is said to* ***contain a substitution*** *if it satisfies one of the following conditions:*

(1) G is a parallel module. Let C_1, ..., C_k be the vertex sets of connected components of G. Then, G is said to be decomposable into C_1, ..., C_k.

(2) G is a series module. Let C_1, ..., C_k be the vertex sets of connected components in the complement of G. Then, G is said to be decomposable into C_1, ..., C_k.

(3) G is a neighborhood module and there exists a module M with $2 \leq |M| < |V(G)|$. Define a module M of G to be a ***maximal submodule*** *if no other proper submodule of N contains M. By [17], the maximal submodules partition the vertex set of N. Thus, G is said to be decomposable to its maximal submodules.*

Such a decomposition is referred to as the ***substitution decomposition*** *or the* ***modular decomposition****. A graph G is said to be* ***s-inseparable*** *if it does not contain a substitution.*

The modular decomposition starts with the module V(G) and creates a corresponding root node in the decomposition tree labeled with S, P or N, depending on the type of the module. Then it decomposes this module into submodules M_1, M_2, ... , M_k using one of the above three rules. For each M_i, find its modular decomposition tree and make its root node a son of the root node representing the module V(G). The resulting tree is called a ***modular decomposition tree*** T of G. In [17], it was shown that there exists a unique modular decomposition tree for a graph G.

Figure 3.1. The substitution decomposition tree of a graph G

The ***representative graph*** of a module M in the decomposition tree T is a graph

induced by a subset consisting of a single vertex from each component submodule of M. We denote the representative graph of each module by G_i and attach it to the root of this module. The representative graph of a series module is a clique. The representative graph of a parallel module is an independent set. The representative graph of a neighborhood module is an s–inseparable graph by our definition.

Definition. *The* ***s–inseparable components*** *of a graph G are defined to be either*

(i) a single vertex when there is no neighborhood module in T, or

(ii) the representative graphs of neighborhood modules in the tree T of G.

The example in Fig. 3.1 has three neighborhood modules and each of their representative graphs is an induced path with four vertices.

Another decomposition we used is the ***join decomposition*** of Cunningham [3].

Definition. *A graph G is said to have a* ***join*** *if V(G) can be partitioned into V_0, V_1, V_2 and V_3 with $|V_0 \cup V_1| \geq 2$ and $|V_2 \cup V_3| \geq 2$ such that every possible edge exists between V_1, V_2 and no edge exists between V_0, $V_2 \cup V_3$ or between $V_0 \cup V_1$, V_3.*

In case G has a join as the above, we say it is decomposable into H_1 and H_2, where H_1 is the induced subgraph $G[V_0 \cup V_1]$ with an extra vertex v_1 (called the ***marker vertex***) adjacent to every vertex in V_1 and H_2 is the induced subgraph $G[V_2 \cup V_3]$ with an extra vertex v_2 adjacent to every vertex in V_2.

A graph G is said to be ***j–inseparable*** if it does not contain any join. A j–inseparable graph with at least four vertices must be connected. Cunningham [3,4] gave an $O(n^3)$ algorithms to decompose a graph completely into j–inseparable components. Gabor et al. [5] later improved it to $O(m \cdot n)$ time. Similar to the modular decomposition, a unique decomposition tree for the join decomposition can be obtained. However, in this case, nodes in the tree are of two kinds: (1) j–inseparable components of G; (2) node Q_i adjacent to i j–inseparable components of G. An example is shown in Fig. 3.2.

Figure 3.2. The join decomposition tree of a graph G

For the special class of interval graphs, we can show that their s–inseparable components have unique interval N–models. For a general circular–arc graph G, we describe the a transformed decomposition in Section 5.

4. The circle graph G_c associated with a circular–arc graph G

Consider a circular–arc graph G. Associate with each arc of an N–model R of G a chord that connects its two endpoints. The resulting chord model is called an ***associated chord model*** D for G. This chord model gives rise to a circle graph (as shown in Figure 4.1). Since two chords in an associated chord model D intersect if and only if their corresponding arcs cross in R, we have the following

Figure 4.1. The chord model associated with a circular–arc model

Lemma 4.1. *Let D_1 and D_2 be any two associated chord models for G. Then they give rise to the same circle graph, which is uniquely determined by the adjacency relationships in G.*

Definition. *Associate with each graph G the graph G_c which has the same vertex set as G such that two vertices in G_c are adjacent if and only if they are strictly but not strongly adjacent in G.*

The following lemma states that, without loss of generality, we may assume that $\overline{G}_c$ is connected.

Lemma 4.2. *Let G be a graph with a disconnected $\overline{G}_c$. Let C_1, C_2, ..., C_k be the components of $\overline{G}_c$. Let H_c^i and H^i, i = 1,..., k, be their corresponding induced subgraphs in G_c and G, respectively. Then G is a circular–arc graph if and only if each of the H^i, i = 1, ..., k, is a circular–arc graph.*

The transformation from an associated chord model to an N–model for G can be

carried out uniquely as shown in the following

Lemma 4.3. *Let G be a circular–arc graph with a connected $\overline{G}_c$. Let D be an associated chord model of G_c. Then there is a unique N–model R of G corresponding to D, namely, the sides of all arcs in R with respect to their associated chords in D are uniquely determined.*

Not every chord model of G_c is an associated chord model for G because it could violate other adjacency requirements in G. In particular, the relationships among "non–crossing" vertices in G give rise to certain "series and parallel" relationships among nonadjacent vertices in G_c. We first define the following relationships for chords in a circle.

Definition. *Three chords $d_1 = (a_1,b_1)$, $d_2 = (a_2,b_2)$ and $d_3 = (a_3,b_3)$ are said to be **in parallel with d_2 between d_1 and d_3** (denoted by $d_1|d_2|d_3$) if the clockwise order of their labels starting with a_1 is $a_1a_2a_3b_3b_2b_1$. On the other hand, d_1, d_2 and d_3 are said to be **in series** (denoted by $d_1–d_2–d_3$) if the clockwise order of their labels starting with a_1 is $a_1b_1a_2b_2a_3b_3$.*

Lemma 4.4. *Let D be any chord model associated with an N–model for G. Let v_i, v_j and v_k be three vertices in G and d_i, d_j, d_k, their corresponding chords in D. Then these three chords satisfy $d_i|d_j|d_k$ if and only if one of the following conditions holds:*

(1) v_k contains v_j; v_j contains v_i.
(2) v_k contains v_j; v_j and v_i cover the circle
(3) v_k is independent of v_j; v_j contains v_i
(4) v_k is independent of v_j; v_j and v_i cover the circle
(5) v_k and v_j cover the circle; v_j is independent of v_i
(6) v_k and v_j cover the circle; v_i contains v_j

Definition. *Three pairwise nonadjacent vertices v_i, v_j and v_k in G_c are said to be **in parallel with v_j between v_i and v_k** (denoted by $v_i|v_j|v_k$) if their relationships in G satisfy one of the six (mutually exclusive) conditions in Lemma 5.4. Three vertices v_i, v_j and v_k are said to be **in parallel** if they are in parallel with one of them "between" the other two. Three pairwise nonadjacent vertices v_i, v_j and v_k of G_c that are not in parallel are said to be **in series** (denoted by $v_i–v_j–v_k$). A set of more than 3 vertices is said to be in parallel (respectively, in series) if every three vertices in the set are in parallel (respectively, in series).*

Below, we describe an efficient method to determine the parallel relationships in G_c. Consider any vertex u of G_c. Let I_u be the set of vertices not adjacent to u in G_c. Partition I_u into two subsets L_u and R_u such that

(a) L_u consists of all vertices in I_u that are either contained in u or strongly adjacent to u in G.
(b) R_u consists of those in I_u that either are not adjacent to u or contain u in G.

These two sets L_u and R_u are referred to as the "***two sides***" of u. It is easy to see that if $v_1 \in L_u$ and $v_2 \in R_u$, then $v_1|u|v_2$. The converse is also true as shown in the following

Lemma 4.5. *If $v_i|v_j|v_k$ in G_c, then (i) v_i and v_k are on different sides of v_j; (ii) v_j and v_k are on the same side of v_i. If $v_i–v_j–v_k$, then v_i and v_k are on the same side of v_j.*

Based on the series and parallel relationships among nonadjacent vertices in G_c, we characterize the chord models associated with an N–model for G, namely, those models of G_c that can be transformed to N–models for G.

Definition. *A model D of G_c is said to be **conformal to G (in short, conformal)** if, for every vertex u of G, the chords associated with vertices in L_u are on one side of d_u (the chord for u) and those in R_u are on the other side of d_u.*

Equivalently, by Lemma 4.4, a model D is conformal if it satisfies that, in G_c,

(4.6) *for any three pairwise nonadjacent vertices v_i, v_j and v_k, $v_i|v_j|v_k$ iff their corresponding chords in D satisfy $d_i|d_j|d_k$.*

Definition. *An arc model R_H for an induced subgraph H of G is an **induced N–model** for H if the arc overlapping relationships in R_H conform to the vertex adjacency relationships in G (rather than those in H). A chord model D_{H_c} for an induced subgraph H_c of G_c is*

conformal *if it satisfies (4.6) for vertices in* H_c. D_{H_c} *is said to be* ***conformal to a vertex u*** *not in* H_c *if there exists a placement for the chord of u into* D_{H_c} *such that the resulting chord model satisfies (4.6).*

We now state the main theorem of this section, which states that associated chord models of G are exactly those that are conformal.

Theorem 4.7. *Let G be a circular–arc graph with a connected* $\overline{G}_c$. *Then a model for* G_c *is conformal if and only if it is a chord model associated with an N–model for G.*

By Theorem 4.7, the problem of testing whether G is a circular–arc graph is equivalent to that of testing whether G_c has a conformal model.

Theorem 4.8. *Consider a circular–arc graph G. If* G_c *is s–inseparable, then G has a unique N–model (or equivalently,* G_c *has a unique conformal model).*

Theorem 4.9. *Let G be a circular–arc graph such that both* $\overline{G}_c$ *and* G_c *are connected. Then each induced subgraph of G corresponding to the representative graph H of a parallel or a neighborhood module M in* G_c *has a unique induced N–model.*

The next section describes the basic framework of our "transformed" decomposition. We shall first illustrate our method on those circular–arc graphs whose associated circle graphs are connected and then describe the reduction from general circular–arc graphs to the above special case in Section 6.

5. Decomposition of a circular–arc graph G with a connected G_c

Our basic decomposition steps can be briefly described in Figure 5.1. Assume G is a circular–arc graph. A pictorial description is given in Figure 5.2.

1. Transform G into a circle graph G_c and construct a conformal model for G_c as follows.
2. For each s–inseparable component H in G_c, construct a unique "conformal" model by further decomposing it into j–inseparable component graphs with unique chord models.
3. Describe a decomposition scheme of G_c, which yields components that not only are s–inseparable, but also satisfy the additional "conformity" requirement (Sections 6 and 7). Show that such a decomposition results in a unique decomposition tree T.
4. Combine the individual conformal models according to T and form a conformal model for G_c, which is then transformed to an N–model for G.

Figure 5.1. Basic Steps in the decomposition algorithm

Figure 5.2. A pictorial description of our transformed decomposition

In this section we apply the transformed decomposition to recognize circular–arc graphs whose associated circle graphs are connected. The main idea is to apply the so called "consistent" decomposition to their associated circle graphs. We assume that the given graph G is indeed a circular–arc graph and proceed with the decomposition. In case G is not a circular–arc graph, our algorithm either detects a contradiction at an intermediate step or, at the end, constructs an arc model for G which violates its vertex adjacency relationships. Throughout this section we assume the graph G satisfies

(5.1) *Both* $\overline{G}_c$ *and* G_c *are connected.*

The general case will be discussed in Section 6.

Our consistent decomposition differs from the modular decomposition only in the very first step of separating the module $V(G_c)$ into its children submodules due to the additional conformity requirements. Let M be any subset of $V(G_c)$. Let I_M denote the set of vertices in $V(G_c)\backslash M$ which are not adjacent to all vertices of M in G_c. Note that M is a

module iff no vertex in I_M is adjacent to any vertex of M.

Definition. *A module M of $V(G_c)$ is consistent with a subset I in I_M if for any two vertices v_1 and v_2 in I, they are on the same side of a vertex u in M if and only if they are on the same side of every vertex in M.*

Consistent modules will be used throughout our consistent decomposition in lieu of modules. By assuming (5.1), $V(G_c)$ must be a neighborhood module. Let M_1, ..., M_k be the maximal submodules which partition $V(G_c)$ in the neighborhood modular decomposition. The M_i's are not necessarily consistent as shown in the example of Figure 5.3. To obtain maximal consistent submodules of $V(G_c)$, we further partition each M_i into its maximal consistent submodules.

Figure 5.3. A maximal submodule of $V(G_c)$ which is not consistent

Lemma 5.2. *Let M_i be a maximal submodule of a neighborhood module $V(G_c)$. Then there is a unique partition of M_i into maximal subsets M^1, M^2, ..., M^r each of which is consistent with I_{M_i}.*

The result in Lemma 5.2 can be strengthened as follows.

Lemma 5.3. *Each M^j, $j = 1, ..., r$, in Lemma 5.2 is consistent (with I_{M^j}) in G_c.*

Definition. *The **consistent partition** of $V(G_c)$ is the unique refinement of maximal submodules of $V(G_c)$ into their maximal consistent submodules as described in Lemma 5.2.*

Lemma 5.4. *Let M be a consistent module of $V(G_c)$ that gives rise to a permutation graph. Then all submodules resulting from the ordinary modular decomposition of G_c are consistent.*

Two submodules M and M' with $M \cap M' = \emptyset$ are said to ***cross*** each other if every vertex in M is adjacent to every one in M'; otherwise, they are said to be ***independent***.

Theorem 5.5. Assume *$V(G_c)$ is a neighborhood module. Let $V' = \{v_1,...,v_r\}$ be a set of representatives from the maximal submodules M^1, ..., M^r in the consistent partition of $V(G_c)$. Then there is a unique conformal model for V'. Furthermore, this model is independent of the v_i's selected (we shall refer to the subgraph induced on V' as the **representative graph for $V(G_c)$** in our consistent decomposition).*

Definition. *The **consistent decomposition tree** T has $V(G_c)$ as its root module, the maximal consistent submodules of $V(G_c)$ as the children submodules of the module $V(G_c)$ and the modular decomposition tree of each children submodule M_i as the subtree of T with root module M_i. Every submodule associated with an internal node of the decomposition tree T is referred to as a **T–module**. The representative graph of every T–module, except possibly $V(G_c)$, is the same as that of its modular decomposition.*

Because the decomposition at each node is unique, the consistent decomposition tree T is unique up to isomorphism. Our method for constructing a conformal model for G_c is to construct conformal models for all T–modules which are also conformal to chords not in these submodules and then to combine them together to form a conformal model for G_c in a bottom–up fashion along the tree T. We state some important properties of this decomposition in Lemma 5.6.

To ensure that a model D_M for a T–module M is conformal to chords not in M we need check conformity of D_M with each vertex in I_M. In Lemma 5.6, we show that it is sufficient to check conformity with just one vertex in I_M. we need more notations to illustrate this.

Since G satisfies (5.1), every T–module except $V(G_c)$ gives rise to a permutation graph. A permutation chord model D_M of a T–module M ($\neq V(G)$) satisfies that there exists a chord d $\notin$ M which intersects every chord in D_M. Hence, there exist two unique minimal segments (a,b) and (a',b') such that every chord in D_M has one label in (a,b), the other label in (a',b'), chord d has one label in (b,a') and the other label in (b',a). we call

segments (a,b) and (a',b') the **main segments** of D_M. By definition, every vertex u in I_M must have its chord placed in either (b,a') or (b',a). Aside from the trivial case that chords in D_M form a clique, D_M must contain a pair of non-intersecting chords $\{v_1,v_2\}$. Then, u, v_1 and v_2 are in parallel and there is a unique placement of the labels of u into (b,a') or (b',a). we use $D_M \oplus u$ to denote the chord model formed by D_M together with the chord u placed properly into (b,a') or (b',a) according to its parallel relationships with chords in D_M.

Lemma 5.6. *Let D_M be a conformal permutation chord model for a* T-module *M of G_c. If $D_M \oplus u$ is conformal for some u in I_M, then $D_M \oplus u'$ is conformal for every u' in I_M.*

Our construction of a conformal model for G_c consists of two steps. In the first step, we construct a conformal model for the representative graph of each T-module which is also conformal to vertices not in the submodule. In the second step, we combine the conformal models of the T-modules in a bottom-up fashion along the tree T.

Theorem 5.7. *For every* T-module *M, the model $D_M \oplus u$ obtained in the above algorithm is conformal for $M \cup \{u\}$.*

Theorem 5.8. *Let G be a graph satisfying (5.1). Let T be the unique consistent decomposition tree for G_c. Then G is a circular-arc graph if and only if every T-module M has a conformal model for its representative graph which is also conformal to every vertex not in M.*

6. Decomposition of a Circular-Arc Graph G with a Disconnected G_c

Consider a circular-arc graph G whose G_c is disconnected. The model construction procedure for each component of G_c becomes more involved because of the complicated series and parallel relationships imposed by other components. Furthermore, to compose a conformal model for G_c, we need arrange the relative positions of its components according to the series and parallel relationships in G. We shall adopt the following approaches:

(1) Record the series and parallel relationships among the components in a unique tree.
(2) Reduce the model construction for each component to that of a circular-arc graph satisfying (5.1).
(3) Combine the individual models to form one for G_c (according to the tree in (1)).

We first discuss the construction of the tree in (1). Let C_1, ..., C_k be the connected components of G_c.

Definition. *Three components C_i, C_j and C_k are said to be* ***in parallel*** *(denoted by $C_i|C_j|C_k$) if there exist three vertices v_i, v_j and v_k from C_i, C_j and C_k, respectively, satisfying $v_i|v_j|v_k$. Three components which are not in parallel are said to be* ***in series*** *(denoted by $C_i-C_j-C_k$) (if $C_i-C_j-C_k$, then any three vertices v_i, v_j and v_k from C_i, C_j and C_k, respectively, must also be in series). A collection of components is in series if every three of them are.*

Definition. *A component C_i is said to be* ***near*** *another component C_j if there exists no component C_k such that $C_i|C_k|C_j$. Define a collection Q of components to be an* ***NS-collection*** *if the components in Q are in series and every two of them are near each other.*

The set of components near a component C_i is unique. Let N_1, N_2, ..., N_r be the maximal NS-collections of G_c.

Lemma 6.1. *The number of maximal NS-collections of G_c is no greater than the number of connected components in G_c.*

Lemma 6.2. *$|N_i \cap N_j| \leq 1$ for $1 \leq i < j \leq r$.*

Definition. *Let C_i be any component of G_c. Let N_j be a maximal NS–collection containing C_i. Define $N_j(C_i)$ to be the collection of those components C such that either (a) $C \in N_j$ or (b) there exists a C' in N_j such that $C|C'|C_i$.*

Lemma 6.3. *Let C_i be any component of G_c. Let N_j be any maximal NS–collection containing C_i. All vertices of all components in $N_j(C_i)$ are on the same side of any vertex of C_i. On the other hand, Let N_j' be any other maximal NS–collection containing C_i. Then, for any two components C, C' of $N_j(C_i)$ and $N_j'(C_i)$, respectively, we have that $C|C_i|C'$. Furthermore, the collection $\{ N_j(C_i) \mid N_j$ contains $C_i \}$ is a partition of the set of components in $G \backslash C_i$.*

Now, construct a bipartite graph T_{NS} with node set on the components C_i's of G_c and the maximal NS–collections N_i's by connecting each node representing a component C_i to every node representing a NS–collection containing C_i.

Theorem 6.4. *The graph T_{NS} is a tree (called the SP–tree of G_c)* (An example is shown in Figure 6.1).

Figure 6.1. The SP–tree T_{NS} of a graph G_c

Definition. *For each component C_i of G_c, define an **expanded component graph** G_i to be the union of C_i together with one vertex from any component of $N_j(C_i)$ for each maximal NS–collection N_j containing C_i.*

We now show that the problem of constructing a conformal model for a disconnected G_c can be reduced to that of constructing a conformal model for each of its expanded component graphs.

Definition. *G_c is said to be **c–inseparable** if there exists a component C_i of G_c whose expanded component graph G_i is the same as G_c.*

Lemma 6.5. *The SP–tree T_{NS} for a c–inseparable graph G_c (= G_i) satisfies that*

(1) every component of G_i except C_i is a leaf in T_{NS} containing only one vertex.

(2) every maximal collection of G_i consists of C_i and a leaf component.

Theorem 6.6. *Suppose G_c is c–inseparable, disconnected and has a conformal model. Let C_i be the component of G_c such that $G_i = G_c$. Let K be the set of vertices of G_c not in C_i. Then any conformal model for G_c satisfies that*

(a) between any two chords u, v not in C_i there is a chord d in C_i such that $u|d|v$.

(b) those chords in K are in series following a unique (up to rotation and reversal) circular order around the circle.

Theorem 6.7. *Assume $\overline{G_c}$ is connected. Then, G_c has a conformal model if and only if each of its expanded component graph G_i, where C_i is a component of G_c, has a conformal model.*

By this theorem, we can assume, without loss of generality, that the given graph G_c is c–inseparable. Let C be the connected component of G_c such that its expanded graph is equal to G_c. Let K be the set of remaining vertices u_1, ..., u_k, each of which constitutes a single–vertex connected component of G_c, and is arranged in a unique circular order according to Theorem 7.6. We need to construct a conformal model for C that also conforms to vertices in K. This problem can be reduced to that of constructing a conformal model for a G_c satisfying (6.1) by adding |K| additional vertices to G_c to make it connected. For each newly added vertex, we specify its series and parallel relationships with respect to all vertices in G_c and the new set of vertices. Such a description follows naturally from Theorems 7.8 and 7.9 below. A module M is said to be ***split*** by a vertex u_j if $u_j \in I_M$ and there exist vertices v, v' in M such that u_j–v–v'. Let T be the consistent decomposition tree of the module C.

Lemma 6.8. *Assume $K = \{u_1\}$. Let M_1, ..., M_r be the children T–submodules of a T–module M in the consistent decomposition. Then, at most one of the M_i's is split by u_1.*

Following Lemma 6.8, we can find a unique path P in tree T from the root (module

C) to a node representing a module M* which is not split by u_1 such that every internal node represents a module split by u_1. Hence, we can make G_c connected by adding a new vertex which connects u_1 to C.

ALGORITHM K(1)

1. If M* is a series module, then add a new vertex u to G_c, make it adjacent to u_1 and M_1, and let all other vertices of G_c be on the same side of u.
2. If M* is a parallel module, then there is a unique linear order among its T–submodule. Consider the unique conformal D of its representative graph. Then, there is a unique segment, say (a,b), to place the chord representing u_1. Let v_i be the vertex in V_{M^*} one of whose chord label is a. Then add a new vertex u to G_c, make it adjacent to u_1 and M_i and all other vertices of G_c be on the same side of u.
3. If M* is a neighborhood module, then there is a unique conformal model D of the representative graph $V_C = \{v_1,...,v_r\}$ of C. The series and parallel relationships dictate that there is a unique segment, say (a,b), to place the chord for u_1. Let v_i be the vertex in V_{M^*} one of whose chord label is a. Then add a new vertex u to G_c, make it adjacent to u_1 and M_i and all other vertices of G_c be on the same side of u.

Call the resulting graph the **K–completion** of G_c.

Definition. *Assume* $|K| \geq 2$. *For* $1 \leq i < j \leq k$, *define* T_{ij} *to be the set of vertices in C which has* $\{u_i,...,u_{j-1}\}$ *on one side and* $K\backslash\{u_i,...,u_{j-1}\}$ *on the other side. Let S be the set of vertices in C each of which is in series with every two vertices in K, namely, the set of vertices in C which are not in any of the* T_{ij}*'s.*

Theorem 6.9. *There is a unique partition of S into* $S_1, S_2, ..., S_k$ *such that in any conformal model D for* G_c, *if the chords* $[a_i,b_i]$ *for* u_i, $i = 1,...,k$, *are clockwise ordered as* $a_1b_1a_2b_2...a_kb_k$, *then the chords in* S_i *have their endpoint labels in segment* (b_i,a_{i+1}).

Definition. *Let* G_c *be a c–inseparable graph . Define its* ***K–completion*** *to be the graph* G_c^* *obtained by adding, for each* u_i *in K,* $i = 1,...,k$, *a vertex* v_i *adjacent to* u_i, u_{i+1} *and all vertices in* T_{ij} *with* $i < j$ *and* T_{mi} *with* $m < i$ (no two v_i's are made adjacent). *Each* v_i *has* S_i *on one side and all other nonadjacent vertices of* G_c^* *on the other side* (an example is shown in Figure 6.7).

Theorem 6.10. *A c–inseparable graph* G_c *has a conformal model if and only if its K–completion graph* G_c^* *has a conformal model.*

7. The Isomorphism Problem on Circular–Arc Graphs

To test the isomorphism between two circular–arc graphs G and G', it suffices to test whether there exist isomorphic conformal models for G_c and G_c'. Our isomorphism algorithm makes use of the unique decomposition tree and the unique conformal models of inseparable components. Note that, a byproduct of our algorithm is an isomorphism algorithm for circle graphs. The latter only involves the join decomposition.

First, consider the isomorphism problem for circular–arc graphs whose G_c satisfy (5.1). We design a iterative algorithm based on their unique consistent decomposition trees T and T'.

Define the ***level*** $\ell(v)$ of a node v in T to be the length (number of edges) of the unique path from v to the root. For each internal node v of T, denote the subtree with root v by T_v. Our isomorphism algorithm (shown in Figure 9.1) assigns labels to the nodes of T and T' such that a node v of T and another node v' of T' receive the same label iff there exist isomorphic conformal models for T_v and T_v'. Denote the set of nodes at level ℓ of T and T' by S_ℓ and S_ℓ', respectively. Let ℓ^* be the largest level number in T. Since each internal node of T has at least two children and the number of leaves is n, the total number of

nodes in T is bounded by O(n), namely, $\sum_{\ell=1}^{\ell^*} |S_\ell| = O(n)$.

ALGORITHM ISO(I)

(1) Assign the label 1 to each leave of T and T'.

(2) Assign labels to the internal nodes of T and T' level by level iteratively with decreasing ℓ. Assume all nodes at level ℓ of T and T' have been assigned labels. Let v and v' be two nodes in $S_{\ell-1}$ and $S'_{\ell-1}$, respectively.

(2.1) If v and v' correspond to series modules in T and T' and their children have the same collections of labels, then assign the same new label to them.

(2.2) If v and v' correspond to parallel modules in T and T' and the labels of their children follow the same unique parallel sequence, then they receive the same new label.

(2.3) If v and v' correspond to neighborhood modules (or, consistent partitions) in T and T', then there are unique conformal models D and D', respectively, for their representative graphs H and H'. We can test isomorphism of v and v' by assigning the chords in D and D' the labels of the respective submodules containing these chords and checking whether the circular label sequences of D and D' are the same (up to rotation and reversal).

The time complexity for (2.3) dominates those for (2.1) and (2.2). Checking the isomorphism between two unique conformal models D and D' takes $O(|D|^2)$ time and the size of D is just the number of children of node v in tree T. Hence, the time it takes to label all nodes in $S_{\ell-1}$ is bounded by $O(|S_\ell|^2)$. Given the tree T and conformal models for all their representative graphs, the overall time complexity for isomorphism testing on connected G_c's is bounded by $\sum_{\ell=1}^{\ell^*} O(|S_\ell|^2) = O(n^2)$, since $\sum_{\ell=1}^{\ell^*} |S_\ell| = O(n)$.

The isomorphism problem for circular–arc graphs whose associated circle graphs are not connected can be solved similarly.

References

1. K.S. Booth and G.S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ–tree algorithms*, **J. Computer System Sci.**, 13 (1976), pp. 335–379.
2. A. Bouchet, *Reducing prime graphs and recognizing circle graphs*, **Combinatorica**, to appear.
3. W.H. Cunningham, *Decomposition of directed graphs*, **SIAM J. Alg. Disc. Meth.**, 3 (1982), pp. 214–228.
4. W.H. Cunningham and J. Edmonds,
5. C.P. Gabor, W.–L. Hsu and K. J. Supowit, *Recognizing circle graphs in polynomial time*, Proc. 26th IEEE Symp. on Foundation of Computer Science, 1985, pp. 106–116.
6. M.R. Garey, D.S. Johnson, G.L. Miller and C.H. Papadimitriou, *The complexity of coloring circular arcs and chords*, **SIAM J. Alg. Disc. Meth.** 1 (1980), pp. 216–227.
7. F. Gavril, *Algorithms on circular–arc graphs*, **Networks**, 4 (1974), pp. 357–369.
8. M.C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 1980.
9. M.C. Golumbic, *Interval graphs and related topics*, **Discrete Math.**, 55 (1985), pp. 113–121.
10. M.C. Golumbic and P.L. Hammer, *Stability in circular–arc graphs*, **J. Algorithms** 9 (1988), 314–320.

11. U.I. Gupta, D.T. Lee and J. Y.–T. Leung, *Efficient algorithms for interval graphs and circular–arc graphs*, **Networks**, 12 (1982), pp. 459–467.
12. V. Klee, What are the intersection graphs of arcs in a circle?, Amer. Math. Monthly 76 (1969), pp. 810–813.
13. W.–L. Hsu, *Maximum weight clique algorithms for circular–arc graphs and circle graphs*, **SIAM J. Comput.**, 14 (1985), pp. 224–231.
14. W.–L. Hsu, *Linear time algorithms for circular–arc graphs*,
15. S. Masuda and K. Nakajima, *An optimal algorithm for finding a maximum in circular–arc graphs*, **SIAM J. Comput.**,
16. F.S. Roberts, *Graph theory and its applications to problems of society*, **SIAM**, Philadelphia, 1978.
17. J. Spinrad, On comparability and permutation graphs, **SIAM J. Comput.**, 14 (1985), pp. 658–670.
18. J. Spinrad,
19. J. Spinrad, *On line modular decomposition*,
20. F.W. Stahl, *Circular genetic maps*, **J. Cell Physiology**, 70 (Suppl. 1) (1967), pp. 1–12.
21. K.E. Stouffers, *Scheduling of traffic lights--A new approach*, **Transportation Res.**, 2 (1968), pp. 199–234.
22. W. Trotter and J. Moore, *Characterization problems for graphs, partially ordered sets, lattice, and families of sets*, **Discrete Math.**, 16 (1976), pp. 361–381.
23. A. Tucker, *Coloring a family of circular–arc graphs*, **SIAM J. Appl. Math.**, 29 (1975), pp 493–502.
24. A. Tucker, *An efficient test for circular–arc graphs*, **SIAM J. Comput.**, 9 (1980), pp 1–24.

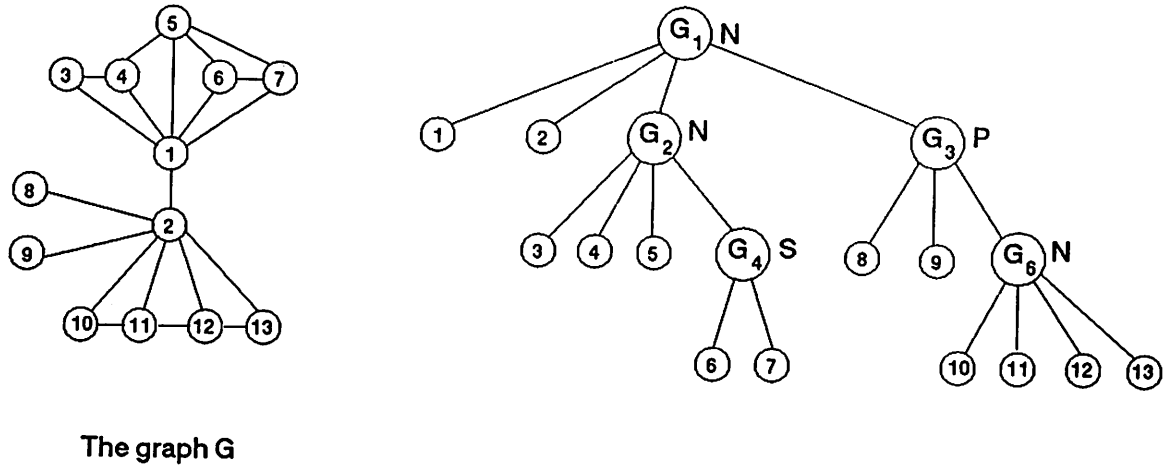


Figure 3.1. The substitution decomposition tree of a graph G

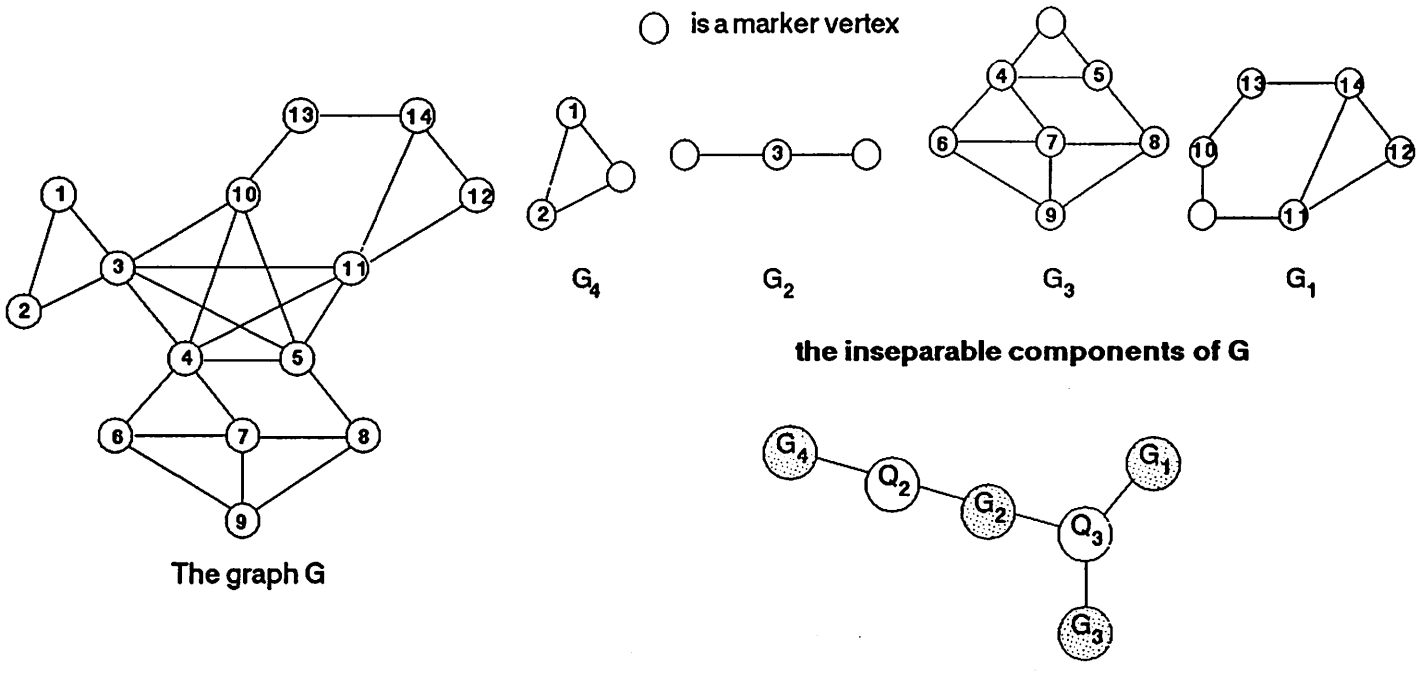
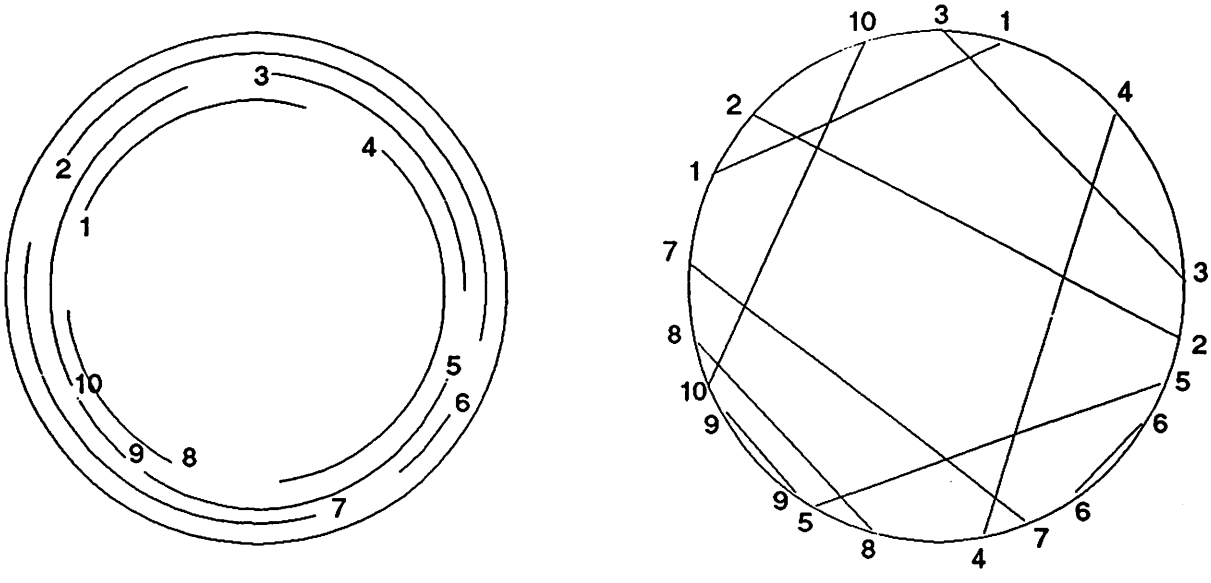


Figure 3.2. The join decomposition tree of a graph G



An arc model for G

Its associated chord model

Figure 4.1. The chord model associated with a circular – arc model

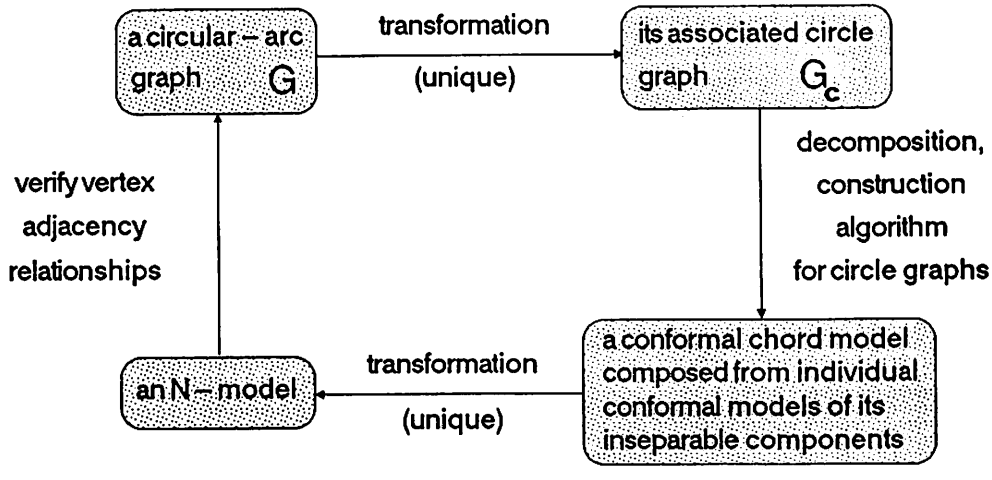
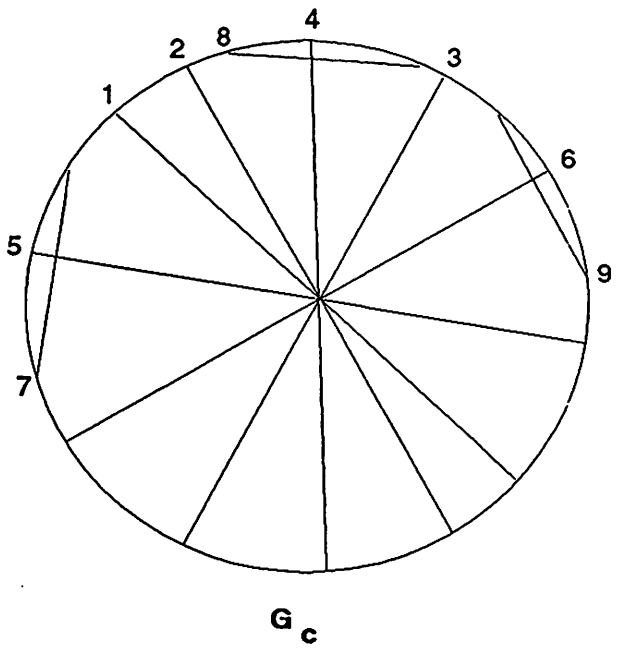


Figure 5.2. The basic framework of the transformed decomposition



The partition of $V(G_c)$ into maximal submodules is

{1,2,3}, {4}, {5}, {6}, {7}, {8}, {9}.

The partition of $V(G_c)$ into maximal consistent submodules is

{1,2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}

Hence the submodule {1,2,3} is not consistent

Figure 5.3. A maximal submodule which is not consistent

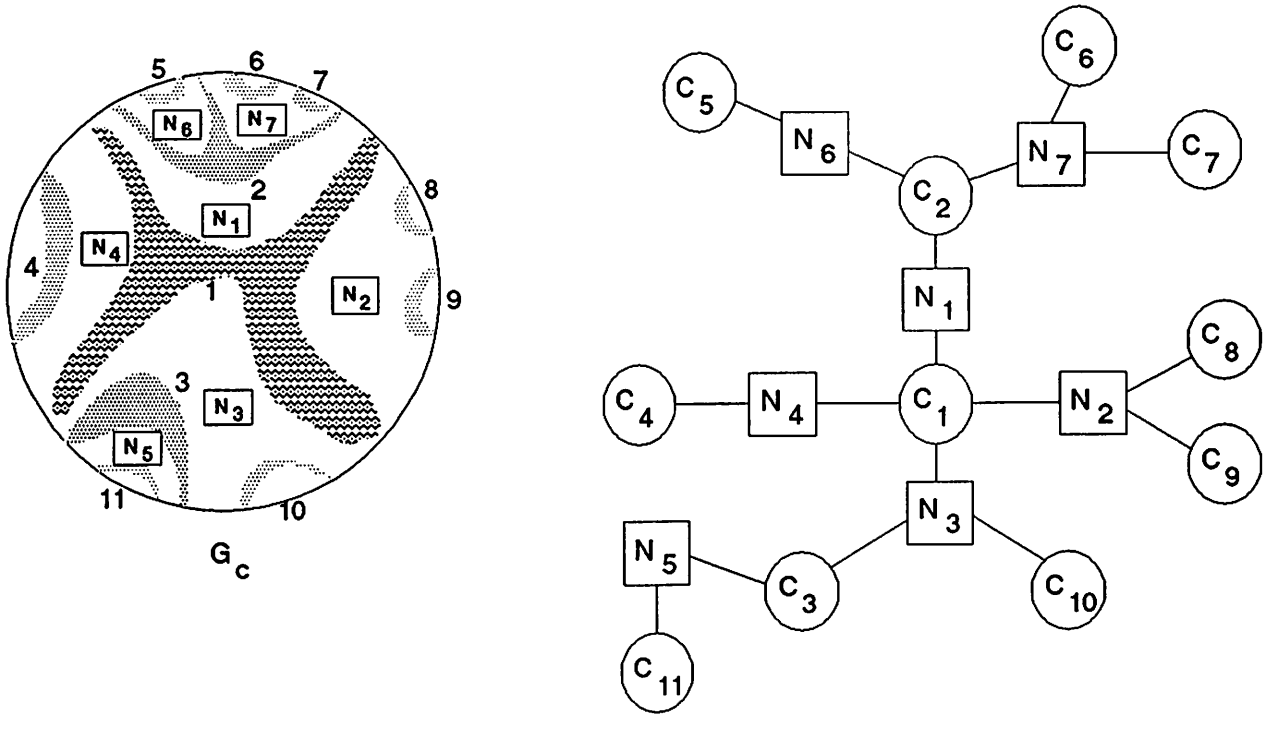


Figure 6.1. The SP – tree T_{NS} of a graph G

Column Generation Methods for Probabilistic Logic

Brigitte Jaumard
GERAD and École Polytechnique de Montréal
5255 avenue Decelles
Montréal, CANADA, H3T1V6

Pierre Hansen
RUTCOR, Rutgers University
Hill Center for the Mathematical Sciences
NJ08903, USA

Marcus Poggi de Aragaö
GERAD and École Polytechnique de Montréal
5255, Avenue Decelles
Montréal, CANADA, H3T1V6

Abstract

Nilsson recently introduced a rigorous semantic generalization of logic in which the truth values of sentences are probability values. This led to state precisely several basic problems of Artificial Intelligence, a paradigm of which is probabilistic satisfiability (PSAT): determine, given a set of clauses and probabilities that these clauses are true, whether these probabilities are consistent. We consider several extensions of this model involving intervals on probability values, conditional probabilities and minimal modifications of probability values to ensure satisfiability. Investigating further an approach of Georgakopoulos, Kavvadias and Papadimitriou, we propose a column generation algorithm which allows to solve exactly all these extensions. Computational experience shows that large problems, with up to 140 variables and 300 clauses, may be solved in reasonable time.

1 Introduction

Expert systems often deal with entailing information from domain and problem knowledge where both the knowledge and the entailment rules are known with some uncertainty. As typical expert systems consist in facts and rules that can be stated as logical sentences, researchers in Artificial Intelligence have been interested in various generalizations of logic where uncertainty is taken into account.

In most early systems such as MYCIN [4] or PROSPECTOR [46], the uncertainty factor has been incorporated in a heuristic fashion. Several formal proposals have also been made, usually diverging from classical probability theory, as for example fuzzy set theory [55], Dempster-Shafer theory [50], or certainty/confidence factor [44] for the most cited. For surveys, see, e.g., Genesereth and Nilsson [13], Leung and Lam [40], Pearl [45], Stephanou and Sage [53].

In 1986, Nilsson [42] proposed a rigorous framework for dealing with the probability of logical sentences from the propositional and first order logic. He followed the usual assumptions of probability theory in postulating that there exists a consistent probability distribution over all possible states in the domain of interest that represents the current knowledge of the decision maker. It describes his information about the relative chances of facts, rules and consequences being jointly true or false. If all probabilities are equal to zero or to one, the framework of probabilistic logic proposed by Nilsson reduces to classical logic.

The main problem addressed by Nilsson [42] is that of the consistency of logical sentences together with their probabilities of being true (Probabilistic Satisfiability or PSAT for short). For instance, assume the sentences *Alice drinks the flask, Alice drinks the flask* $\Longrightarrow$ *Alice shrinks*, and

Alice shrinks have probabilities 0.8, 0.6 and 0.3. Are these probabilities consistent ? More generally, Nilsson's probabilistic logic allows to evaluate the probability that new expressions such as *Alice drinks the flask* $\Longrightarrow$ *Alice swells* are true (Probabilistic Entailment, the optimization version of PSAT), the result being an interval of values possibly reduced to a point.

Nilsson [42] shows that PSAT can be expressed as a large linear program, with a number of columns exponential in the number of elementary facts. He proposes to solve small instances by the simplex method and large ones by heuristics.

Many authors comment on Nilsson's paper. Grosof [19] discusses various extensions of Nilsson's and other models, based on a logic of upper and lower bounds on conditional probabilities, without considering algorithmic and implementation issues. Dubois and Prade [11] [12], Horvitz and Heckerman [28], Paass [43] and Reiter [47] compare Nilsson's model with other approaches to reasoning under uncertainty. Chen [7], Grosof [20] and McLeish [39] propose to extend the model to deal with nonmonotonic logic, in which inconsistencies may be detected and lead to revision of rules or probabilities.

Algorithmic and complexity issues are addressed in detail in two papers, respectively by Georgakopoulos, Kavvadias and Papadimitriou [14] and by Kavvadias and Papadimitriou [31]. They first show that PSAT is NP-complete and remains so even when all clauses contain at most two variables (2PSAT). Then, they propose to solve PSAT by column generation and prove that the selection of the entering column is equivalent to a weighted maximum satisfiability (MAXSAT) problem. This last problem being NP-complete, Kavvadias and Papadimitriou [31] propose a heuristic to solve it. Their algorithm for PSAT thus remains heuristic in that it is not proved that no column with positive reduced cost remains when the heuristic cannot find any more. They solve problems with up to 50 variables and 80 clauses.

In this paper, we extend Nilsson's model and the approach of Georgakopoulos, Kavvadias and Papadimitriou [14] [31] as follows: after summarizing Nilsson's model in the next Section, we show that extensions of it allow to take into account intervals of probability values and conditional probabilities to be true for the sentences. Moreover, we show that it can be modified to determine minimum changes in the probability values or intervals which restore consistency in the case where the initial probabilities are inconsistent. In Section 3, we present an exact column generation algorithm which allows to solve all these problems. Selection of the entering column is done first by a heuristic method of Tabu Search type, and then by an exact algorithm of pseudo-Boolean programming, when no more column with positive reduced cost can be found with the heuristic. This algorithm provides a positive answer to a question of Georgakopoulos, Kavvadias and Papadimitriou [14] about the possibility of solving probabilistic logic problems with conditional probabilities (CONDSAT problems) by column generation. Extensive computational experience is reported on in Section 4. Problems with up to 140 variables and 300 clauses with similar characteristics to those used in typical expert systems are solved in reasonable time. Conclusions are drawn in the last Section.

2 Probabilistic Logic Problems

2.1 Nilsson's model and PSAT

Consider a set of m logical sentences $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ defined on a set of n propositional (boolean) variables $X = \{x_1, x_2, ..., x_n\}$. These variables correspond to elementary facts, which are *true* or *false*. A literal y_k is any propositional variable ($y_k = x_k$) or its complement ($y_k = \bar{x}_k$). For ease of exposition, we consider sentences from the propositional calculus. Nilsson's model

applies also to first order logic. Standard techniques to reduce first order logic to propositional logic are described in Chang and Lee [6]. We then assume, without loss of generality, that as in most expert systems, sentences S_i are logical implications of the form:

$$S_i: \quad \varphi_i \Longrightarrow \psi_i$$

where φ_i and ψ_i are boolean functions; φ_i is called the *antecedent* (or premise) of the implication and ψ_i the *consequent.*

The antecedent φ_i can be written in Disjunctive Normal Form (DNF), i.e., as the disjunction of conjunctions of literals, and the consequent ψ_i in Conjunctive Normal Form (CNF), i.e., as the conjunction of disjunctions of literals. Moreover, in propositional calculus, the logical implication:

$$\varphi_i^1 \vee \varphi_i^2 \Longrightarrow \psi_i$$

is equivalent to the two elementary implications:

$$\varphi_i^1 \Longrightarrow \psi_i \quad \text{and} \quad \varphi_i^2 \Longrightarrow \psi_i.$$

Similarly,

$$\varphi_i \Longrightarrow \psi_i^1 \wedge \psi_i^2$$

is equivalent to:

$$\varphi_i \Longrightarrow \psi_i^1 \quad \text{and} \quad \varphi_i \Longrightarrow \psi_i^2.$$

This decomposition is often used in expert systems, where sentences have the form:

$$\bigwedge_{j \in A} y_j \Longrightarrow \bigvee_{j \in C} y_j$$

with a few literals on each side and often a single one on the right-hand side (see, e.g., MYCIN [4]). Note that such elementary implications correspond to disjunctions of literals, i.e., *clauses*:

$$\left(\bigvee_{j \in A} \overline{y}_j\right) \bigvee \left(\bigvee_{j \in C} y_j\right). \tag{1}$$

The fact that in propositional calculus logical implications can be reduced to sets of clauses does not carry over to probabilistic logic. Indeed, there does not appear to be any way to deduce from the probability π_i that S_i is true, truth probabilities for the clauses (1) which are logically equivalent to S_i. It is therefore of interest to extend Nilsson's model to the case where the S_i are logical sentences which are not necessarily clauses. It will be shown below that this generalization does not substantially change the proposed solution procedure.

Nilsson [42] defines a *world* as any truth assignment w over $\mathcal{S}$. A world w is *possible* if there exists a truth assignment over X which leads to w over $\mathcal{S}$, and the world is *impossible* otherwise. In fact, a possible world can be equivalently defined as the set of truth assignments over X which lead to the same truth assignment w over $\mathcal{S}$. Let W denote the set of possible worlds. From now on, we will deal only with possible worlds.

Let $p = (p_1, p_2, ...)$ be a *probability distribution* on W. Assume we are also given m probabilities $\pi_1, \pi_2, ..., \pi_m$, one for each logical sentence. The probability distribution *satisfies* the set of logical sentences together with their probabilities if: *for each sentence* S_i $(i = 1, 2, ..., m)$, *the sum of all* p_j*'s over all truth assignments* w_j *which satisfy* S_i *equals* π_i.

Let A be an $m.|W|$ matrix such that a_{ij} is equal to 1 if w_j satisfies S_i, and equal to 0 otherwise.

The *Probabilistic Satisfiability* (PSAT) problem is defined as follows:

Is there a probability distribution p such that the system

$$\begin{cases} \mathbb{1}.p & = 1 \\ Ap & = \pi \\ p & \geq 0 \end{cases} \tag{2}$$

admits at least one solution?

Example 1. Consider the example of the introduction, with $x_1 \equiv$ *Alice drinks the flask*, $x_2 \equiv$ *Alice shrinks* and $\mathcal{S} = \{x_1, x_1 \Longrightarrow x_2, x_2\}$. $\mathcal{S}$ leads to four possible worlds (1,1,1), (1,0,0), (0,1,1) and (0,1,0) (which correspond to the truth assignments over X, i.e., (1,1), (1,0), (0,1) and (0,0)). PSAT is then to find p such that:

$$\begin{cases} p_1 + p_2 + p_3 + p_4 = 1 & (S_0) \\ p_1 + p_2 = 0.8 & (S_1 : x_1) \\ p_1 + p_3 + p_4 = 0.6 & (S_2 : x_1 \Longrightarrow x_2) \\ p_1 + p_3 = 0.3 & (S_3 : x_2) \\ p_j \geq 0 & j = 1,2,3,4, \end{cases}$$

or show that there is no such p (which is indeed the case).

Assume now that the answer to the PSAT problem defined by $\mathcal{S}$ and π is positive, or, for short, that $(\mathcal{S}, \pi)$ is consistent. Let S_{m+1} denote an additional logical sentence, possibly deduced from $\mathcal{S}$. The *Probabilistic Entailment* problem of Nilsson [42] is to determine the range $[\underline{\pi}_{m+1}, \overline{\pi}_{m+1}]$ of values of the probability π_{m+1} associated with S_{m+1} such that $(\mathcal{S} \cup \{S_{m+1}\}, (\pi, \pi_{m+1}))$ is consistent. This problem can be solved by considering an objective function $A_{m+1}p$, where $a_{m+1,j}$ is equal to 1 if S_{m+1} is true for the possible world w_j and equal to 0 otherwise, and determining $\underline{\pi}_{m+1} = \min A_{m+1}p$ and $\overline{\pi}_{m+1} = \max A_{m+1}p$ subject to the constraints (2). Using the terminology of computational complexity, PSAT is a decision problem, corresponding to the optimization problem Probabilistic Entailment (or Probabilistic Entailment is the optimization form of PSAT).

Note that S_{m+1} may involve variables not appearing in the logical sentences of $\mathcal{S}$. Moreover, even if this is not the case, the set of possible worlds is modified, as they have one more component. Their number is between once and twice the number of worlds before introducing S_{m+1}.

Example 2. Consider again Example 1, setting π_3 to 0.5 to obtain consistency. Add the logical sentence $S_3 \equiv$ *Alice drinks the flask* $\Longrightarrow$ *Alice swells*, i.e., $x_1 \Longrightarrow x_3$ with $x_3 \equiv$ *Alice swells*. There are now six possible worlds, i.e., (1,1,1,1), (1,1,0,0), (1,0,1,1), (1,0,1,0), (0,1,1,1) and (0,1,0,0). The third one corresponds to the truth assignments (0,1,1) and (0,1,0) over X and the fourth one to the truth assignments (0,0,1) and (0,0,0).

Min (Max) $\pi_4 = p_1 + p_2 + p_3 + p_4$

subject to:

$$\begin{cases} p_1 + p_2 + p_3 + p_4 + p_5 + p_6 = 1 \\ p_1 + p_2 + p_5 + p_6 = 0.8 \\ p_1 + p_3 + p_4 + p_5 = 0.6 \\ p_1 + p_3 + p_5 = 0.5 \\ p \geq 0 \end{cases}$$

yields $[\underline{\pi}_4, \overline{\pi}_4] = [0.2, 1]$

Nilsson's [42] model can also be used to derive, by geometric or algebraic means, known and new probability formulae. For instance, Nilsson [42] considers the system S = $\{x_1, x_1 \Longrightarrow x_2\}$ and obtains the following bounds on $\pi(x_2)$:

$$\pi(x_1) + \pi(x_1 \Longrightarrow x_2) - 1 \quad \leq \quad \pi(x_2) \quad \leq \quad \pi(x_1 \Longrightarrow x_2)$$

already derived by Suppes [54].

2.2 PSAT with interval and conditional probabilities

It may be more relevant in order to represent uncertainty on the truth of the logical sentences to associate probability intervals with them rather than a single probability value. Then, the width of the interval represents the confidence of the expert in the certainty measure. Such interval - valued models have attracted interest in Artificial Intelligence, due to Shafer [50] and to more recent work that attempts to make Shafer's theory practically useful (e.g., Gordon and Shortliffe [18]).

The PSAT problem readily extends to the case where probability intervals are given for the truth of sentences. It is then defined as follows:

Is there a probability distribution p such that the system

$$\begin{cases} \mathbb{1}.p = 1 \\ \underline{\pi} \leq Ap \leq \overline{\pi} \\ p \geq 0 \end{cases} \tag{3}$$

admits at least one solution ?

The optimization version of this problem is obtained by adding to (3) the objective function of Probabilistic Entailment.

Examples 3 and 3'. Consider again Example 1. The probability interval $[\underline{\pi}_3, \overline{\pi}_3]$ entailed by $(\{S_1, S_2\}, (0.8; 0.6))$ is equal to $[0.4; 0.6]$. Assume now that $\underline{\pi}_1 = 0.7$, $\overline{\pi}_1 = 0.9$, $\underline{\pi}_2 = 0.5$ and $\overline{\pi}_2 = 0.7$. The entailed interval $[\underline{\pi}_3, \overline{\pi}_3]$ is then $[0.2; 0.7]$.

Many expert systems make use of knowledge which is known with sufficient precision only in some situations. It is usually expressed with conditional probabilities. For instance, given the sentences $S_1 \equiv$ *Alice drinks the flask* and $S_3 \equiv$ *Alice shrinks*, the conditional probability $\pi(S_1|S_3)$ is that fraction of cases in which S_3 is true for which S_1 also is true. Note that it is easier to evaluate $\pi(S_1|S_3)$ than $\pi(S_1 \Longrightarrow S_3)$ as the latter is equal to the fraction of cases in which S_1 is false plus the fraction of cases in which both S_1 and S_3 are true. Thus in order to evaluate $\pi(S_1 \Longrightarrow S_3)$, one needs knowledge of how often both S_1 and S_3 are false, which might be difficult to obtain and which is not necessary for computing $\pi(S_1|S_3)$.

Nilsson's model [42] can be modified as follows to allow simultaneous treatment of simple and conditional probabilities.

We consider again a set $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m logical sentences. Only the simple probabilities for the first $q < m$ of them are known. Let $Q = \{1, 2, ..., q\}$. The $m - q$ last sentences appear, with possibly some or all of the first q sentences, in conditional probabilities $\pi_{i/j} = \pi(S_i|S_j)_{(i,j)\in T}$ where $T \subseteq M \times M$ and $M = \{1, 2, ..., m\}$. Let $C \subseteq M$ be the index set of the sentences appearing as condition in the probabilities $\pi_{i/j}$. By definition,

$$\pi_{i/j} = \pi(S_i|S_j) = \frac{\pi(S_i \cap S_j)}{\pi(S_j)} = \frac{(A_i \wedge A_j)p}{\pi_j} \tag{4}$$

where $A_i \wedge A_j = (a_{ik} \wedge a_{jk})_k$.

Then, the *Conditional Probabilistic Satisfiability* (CONDSAT) problem can be expressed:

Problem CONDSAT

Is there a probability distribution p such that:

$$\begin{cases} \mathbb{1}.p & = 1 & \\ A_i p & = \pi_i & i \in Q \\ A_j p - \pi_j & = 0 & j \in C \cap (M \backslash Q) \\ (A_i \wedge A_j).p - \pi_{i/j}\pi_j & = 0 & (i,j) \in T \\ \pi_j \geq 0 & & j \in C \cap (M \backslash Q) \\ p \geq 0 & & \end{cases}$$

admits at least one solution ?

An optimization version of CONDSAT is obtained by adding an objective function as done for PSAT. One can also use a conditional probability $\pi_{i/j}$ as objective function. In that case, if the denominator π_j is unknown, CONDSAT is an hyperbolic linear program. Nilsson [42] and Paass [43] propose a heuristic method to solve problems of this form. An exact solution can be obtained by solving a sequence of linear programs, following a standard technique of hyperbolic programming (see, e.g., Isbell and Marlow [29], Schaible and Ibaraki [49]). Consider a linear program in which the objective funtion $\pi_{i/j} = A_{m+1}p$ is replaced by $(A_i \wedge A_j)p - \lambda\pi_j$ where λ is a parameter. An optimal solution to the hyperbolic program is obtained for the value λ^* of λ for which the new objective function is equal to zero, and the corresponding optimal value is λ^*. So a first value of λ for which the new objective function is positive at the optimum is chosen (1 for instance, assuming we are maximizing). If the optimal value is strictly negative, λ is set equal to the optimal value of $\frac{(A_i \wedge A_j)p}{\pi_{i/j}}$ and the procedure is iterated.

Again probability intervals can be used instead of probability values. This leads to the most general case we address in this paper.

Determine $\underline{\pi}_{m+1}, \overline{\pi}_{m+1}$ such that:

$$\underline{\pi}_{m+1} = \min A_{m+1}p$$

subject to:

$$\begin{cases} & \mathbb{1}.p & = 1 & \\ \underline{\pi}_i \leq & A_i p & \leq \overline{\pi}_i & i \in Q \\ & A_j p - \pi_j & = 0 & j \in C \cap (M \backslash Q) \\ & (A_i \wedge A_j).p - \overline{\pi}_{i/j}\pi_j & \leq 0 & (i,j) \in T \\ 0 \leq & (A_i \wedge A_j).p - \underline{\pi}_{i/j}\pi_j & & (i,j) \in T \\ & \pi_j \geq 0 & & j \in C \cap (M \backslash Q) \\ & p \geq 0 & & \end{cases}$$

and

$$\overline{\pi}_{m+1} = \max A_{m+1}p \quad \text{subject to the same set of constraints.}$$

2.3 Inconsistency elimination

Many expert systems are designed in such a way that on-line updating of the rules can be easily done. This often entails redundancy as well as inconsistency in the rule database. Inconsistency of a PSAT problem implies that the truth probabilities of some sentences have been incorrectly assessed by the expert(s) who provided them. This is not surprizing as these probabilities are usually subjective and asking for a precise value may be forcing the expert to express knowledge of a more precise nature than that he possesses (this problem is less acute when intervals for

probability values are used). One would thus want to attain consistency with minimal changes following the *nonmonotonic logic* paradigm of Artificial Intelligence. The easiest way to do this is by revising the truth probabilities (an alternate way is to delete a minimum set of sentences). Usually, the expert will be quite sure that some probabilities are close to be exact while he will have doubts about other ones having much accuracy (cf discussions of malfunction diagnosis in chemical plants [51], medical diagnosis of infectious diseases in MYCIN [4] and accident diagnosis for nuclear reactors [41]). This may be captured by assuming he can specify, together with the bounds $\underline{\pi}_i$ and $\overline{\pi}_i$, *attenuation factors* $\underline{w}_i$ and $\overline{w}_i$ the larger the more he believes the interval of the probability values π_i to be accurate. One may then seek probability vectors $\underline{\pi}' = (\underline{\pi}'_1, \underline{\pi}'_2, ..., \underline{\pi}'_m)$ and $\overline{\pi}' = (\overline{\pi}'_1, \overline{\pi}'_2, ..., \overline{\pi}'_m)$ such that the PSAT problem $(\mathcal{S}, [\underline{\pi}', \overline{\pi}'])$ is consistent and the (weighted) distance between $[\underline{\pi}, \overline{\pi}]$ and $[\underline{\pi}', \overline{\pi}']$ is minimum. Using the L_1 norm then yields the following linear program:

$$\text{Minimize} \quad \sum_{i=1}^{m} (\underline{w}_i \ell_i + \overline{w}_i u_i)$$

subject to:

$$\begin{cases} \mathbb{1}.p = 1 \\ \underline{\pi}_i - \ell_i \leq A_i p \leq \overline{\pi}_i + u_i \qquad i \in M \\ p \geq 0 \end{cases}$$

We call this model *Restore Satisfiability* (RSAT).

Again extension to the case of conditional probabilities is straightforward.

Example 4. Consider again Example 1, an inconsistent instance of PSAT. Assume $\underline{w}_1 = \overline{w}_1 = 1$ (Alice's behavior being somewhat impredictible), $\underline{w}_2 = \overline{w}_2 = 10$, $\underline{w}_3 = \overline{w}_3 = 2$. Then the optimal solution to RSAT is $\ell_1^* = 0$, $u_1^* = 0$, $\ell_2^* = 0$, $u_2^* = 0$, $\ell_3^* = 0$, $u_3^* = 0.1$, i.e., $\pi_1 = 0.8$, $\pi_2 = 0.6$ and $\pi_3 = 0.4$.

3 Column generation method

3.1 Definition of the column generation subproblem

The linear programs for the PSAT, CONDSAT and RSAT problems (in their decision or optimization form, and possibly with probability intervals) have an exponential number of columns. These columns can be partitioned into:
- *explicit* columns associated with: (i) slack and artificial variables and (ii) unknown probabilities of logical sentences arising in the expression of conditional probabilities;
- *implicit* columns associated with the probabilities of possible worlds.
This allows us to use the so-called *column generation* technique (Gilmore and Gomory [15], Chvatal [8]).

Let $\alpha = (\alpha_{ij})$ denote the matrix of coefficients of the generic subproblem (PSAT, CONDSAT or RSAT). For simplicity of notation we assume the number of constraints to be $m+1$. We also assume the optimization problem is a maximization one.

Recall that any double sided-inequality $\underline{\pi}_i \leq A_i p \leq \overline{\pi}_i$ can be reduced to an equality with a single bounded slack variable $A_i p - s = \underline{\pi}_i$ where $0 \leq s \leq \overline{\pi}_i - \underline{\pi}_i$. Thus problems with probability intervals will not have more explicit constraints than similar problems with probability values.

The *subproblem* to be solved at each iteration, i.e., generation of a column with a positive reduced cost, corresponds (as we now show) to an unconstrained nonlinear 0–1 program for implicit columns and to the scanning of reduced costs for explicit columns. Consider first the implicit columns. Each logical sentence S_i is expressed as a logical implication $\varphi_i \Longrightarrow \psi_i$ or equivalently $\overline{\varphi}_i \vee \psi_i$. Recall that any Boolean expression can be converted to an equivalent arithmetic expression, by associating *true* and *false* with 1 and 0 respectively and using the identities $\varphi \vee \psi = \varphi + \psi - \varphi\psi$ and $\varphi \wedge \psi = \varphi\psi$. Thus $\overline{\varphi}_i \vee \psi_i$ is equivalent to the expression $\overline{\varphi}_i + \psi_i - \overline{\varphi}_i\psi_i$ which reduces to $1-\varphi_i\overline{\psi}_i$ using the identities $\varphi_i+\overline{\varphi}_i = 1$ and $\psi_i+\overline{\psi}_i = 1$. If $\overline{\varphi}_i$ and ψ_i are literals $1-\varphi_i\overline{\psi}_i$ is an arithmetic expression in 0–1 variables (or a pseudo-Boolean function). If $\overline{\varphi}_i$ and ψ_i are general logical expressions, $1-\varphi_i\overline{\psi}_i$ can be reduced to an arithmetic expression in 0–1 variables by further applications of the identities stated above. The subproblem for implicit columns can therefore be expressed as:

$$\max_j \quad \left(\overline{c}_j = c_j - v_0 - \sum_{i=1}^{m} \alpha_{ij} v_i\right)$$

where v_0 and v_i for $i = 1, 2, ..., m$ are the current simplex multipliers (or dual variables), or

$$\max_j \quad c_j - v_0 - \sum_{i=1}^{m} \left(1 - \varphi_i\overline{\psi}_i\right) v_i, \tag{5}$$

as $\alpha_{ij} = 1$ if and only if $\varphi_i \Longrightarrow \psi_i$ is true, i.e., $\varphi_i\overline{\psi}_i = 0$.

When sentences are reducible to clauses, i.e., disjunctions of literals (see Section 1) this last expression is equal to:

$$\max_y \quad z = c_j - v_0 - \sum_{i=1}^{m} v_i \left(1 - \prod_{j\in A_i} y_j \prod_{j\in C_i} \overline{y}_j\right)$$

or,

$$\max_y \quad z = c_j - \sum_{i=0}^{m} v_i + \sum_{i=1}^{m} v_i \prod_{j\in A_i} y_j \prod_{j\in C_i} \overline{y}_j. \tag{6}$$

For general logical sentences, expression (5) can be written in the arithmetic form

$$\max_y \quad z = c_j - \sum_{i=0}^{m} v_i + \sum_{i=1}^{\mu} v'_i \prod_{j\in L_i^+} y_j \prod_{j\in L_i^-} \overline{y}_j \tag{7}$$

where the v'_i are the coefficients of the μ terms involving 0–1 variables and are equal to sums or differences of dual variables, and L_i^+ (resp. L_i^-) is the index set of the positive (resp. negative) literals.

Then, to solve the subproblem, the value of the optimal solution of this unconstrained nonlinear 0–1 program must be compared to the largest reduced cost in an explicit column, and the column corresponding to the maximum of these values selected. If this column is an explicit one, it is immediately available; if it is an implicit one, the corresponding column in the original problem is obtained by setting $\alpha_{ij} = 1$ if and only if $\varphi_i\overline{\psi}_i = 0$ in the optimal solution of (6) or (7) and the column in the current tableau is generated by multiplying this original column by the inverse of the current basis. Except for the way this subproblem is treated, the algorithm to solve the generic subproblem is the standard column generation method for linear programming (A very clear exposition of this method is given in Chapter 18 of Chvatal [8]). We next discuss how to maximize (6) or (7).

3.2 Resolution of the column generation subproblem

At each iteration of the column generation method a column with positive reduced cost must be found by a heuristic or an exact algorithm. Optimization problems require finding the largest positive (or smallest negative) reduced cost at least once, i.e., at the last iteration, if the algorithm is to guarantee an optimal solution has been found. Decision problems are solved as soon as a feasible solution is found, but finding none when choosing the entering column in a heuristic way does not guarantee there are none. Kavvadias and Papadimitriou [31] use a *variable depth local search* heuristic to maximize (6), see Lin and Kernighan [36] [37]. We use a *Steepest Ascent Mildest Descent* (SAMD) or *Tabu search* heuristic (Hansen and Jaumard [24] [30], Glover [16] [17]) to find an approximately optimal solution and a variant of the *Basic algorithm* of Hammer, Rosenberg and Rudeanu [22] [23], the *Basic algorithm revisited* of Crama, Hansen and Jaumard [10], to find an exact optimal solution of (6) or (7). The latter algorithm being explained elsewhere, we describe only the former. SAMD proceeds to a local optimum by switching one variable at a time along a direction of steepest ascent, then leaves this local optimum along a direction of mildest descent and avoids cycling (as far as possible) by forbidding for some time ascent moves which are the reverse of descent ones. Its rules are the following:

Procedure SAMD
Logical change;
Select an initial vector $\tilde{y}$ and compute $z(\tilde{y})$;
Set $f_j = 0$ for $j = 1, \ldots, n$;
Change $\longleftarrow$ true; $z^* \longleftarrow z(\tilde{y})$; $y^* \longleftarrow \tilde{y}$;
While change do
  Change $\longleftarrow$ false;
  Repeat rep **times**
    Transform $\tilde{y}$ into $\tilde{y}^j$ by switching variable $\tilde{y}_j$;
    Evaluate $\delta_j = z(\tilde{y}^j) - z(\tilde{y})$ for $j = 1, 2, \ldots, n$ such that $f_j = 0$;
    Let $\delta_k = \max \{\delta_j | j = 1, 2, \ldots, n \text{ and } f_j = 0\}$;
    $\tilde{y} \longleftarrow \tilde{y}^k$;
    If $\delta_k \leq 0$ then $f_k \longleftarrow t$;
    If $z(\tilde{y}) > z^*$ then $z^* \longleftarrow z(\tilde{y})$;
      $y^* \longleftarrow \tilde{y}$;
      change $\longleftarrow$. true . ;
    $f_j \longleftarrow f_j - 1$ for all $j = 1, 2, \ldots, n$ such that $f_j \neq 0$
  End repeat
Endwhile

The logical variable *change* is used to check if a better solution than previously known is found in a cycle of *rep* iterations. The incumbent solution is kept in y^* and its value in z^*. The parameters f_j denote the remaining number of iterations (or *tabu factor* initialized to t) during which a switch of variable y_j is forbidden. Note that the δ_j are the first order partial derivatives of z.

To allow an efficient implementation, the function is represented with a double chaining, one on literals in terms and another one on terms in which a given variable appears. Updating is then used to reduce computation time per iteration in SAMD. During initialization, the numbers of literals which are true in each term are computed. These numbers are updated after each switching of variable using the chaining on terms containing y_k and $\overline{y}_k$. The δ_j are initialized at the sum of coefficients of terms containing y_j all literals of which are true except possibly for y_j, minus the sum of coefficients of terms containing $\overline{y}_j$ all literals of which are true except possibly for $\overline{y_j}$. If, after switching, literal y_k or $\overline{y}_k$ becomes false in a term and is the only false one in that term, the

chaining on variables is used to update the δ_j for which y_j or $\overline{y}_j$ $(j \neq k)$ belongs to this term by subtracting the coefficient of the term. Similar procedures are used for updating when the number of false literals in a term goes from 2 to 1 or from 1 to 0. Then δ_k is updated by multiplying it by -1. In this way the number of computations per iteration is in the order of the number of literals in (7) in worst case, and much smaller on average for problems with short sentences which are common in expert systems.

The heuristic of Kavvadias and Papadimitriou [31] is close to the one we use (which was developed independently [24], [30]). At each iteration, their variable depth local search algorithm explores a sequence of n solutions obtained by switching one variable at a time, i.e., that one which increases z the most (or decreases it the least) and forbidding the reverse move. Then it goes to the best solution among them provided the latter improves upon the incumbent value. If none does, the algorithm stops. This last rule is more rigid than in SAMD, in which the parameter *rep* is usually larger then n. SAMD may therefore explore more thoroughly the set of solutions of (6) or (7) but at a higher computational cost.

3.3 Polynomial cases

Georgakopoulos, Kavvadias and Papadimitriou [14] show that PSAT is in NP. It is clearly NP-complete, as it contains SAT as a particular case. They show it remains NP-complete in the particular case called 2PSAT, i.e., when the logical sentences are clauses with at most two literals. Moreover, they initiate the study of polynomial cases of PSAT. To this effect, they consider the resolution of the dual of PSAT by Khachiyan's [32] algorithm, keeping the exponential length list of inequality constraints implicit. Generating a violated inequality is equivalent to a weighted MAXSAT problem. In such a problem m clauses on n variables are given, together with weights and it is asked whether there is a truth assignment such that the sum of weights of all satisfied clauses is larger than or equal to a given value. They prove that an instance of PSAT with m clauses and n variables can be solved in $O(m^2 log m)$ iterations, each of which involves solving an instance of WEIGHTED MAXSAT on the same clauses (with weights of $O(m)$ bits length) and performing $O(m^3 log m)$ more operations.

Thus the problem of finding polynomial cases of PSAT reduces to that of finding such cases for WEIGHTED MAXSAT. For a given set of clauses on n variables, the *co-occurrence graph* [10] has vertices associated with the variables and edges joining pairs of vertices if and only if the variables (negated or not) associated with their endpoints appear together in a clause. Recall a graph is *outerplanar* if it is planar and can be embedded in the plane so that all of its vertices lie on the same face. Georgakopoulos, Kavvadias and Papadimitriou [14] show that PSAT problems with outerplanar co-occurrence graph and at most two literals per clause can be solved in polynomial time. This implies as a corollary that PSAT problems with acyclic co-occurrence graph are polynomially solvable.

Kavvadias and Papadimitriou [31] provide further complexity results: they first show that two particular cases of 2PSAT are NP-complete: PLANAR 2PSAT and COMPATIBLE MARGINALS, i.e., the problem of finding if probability distributions for all four conjunctions for several pairs of variables are compatible. Then they prove that finding a column with positive reduced cost in the solution of PSAT (in its decision form) by phase 1 of the simplex algorithm, a problem they call PIVOT MAXSAT, is also NP-complete.

In fact, the WEIGHTED MAXSAT problem has long been studied under the name of unconstrained nonlinear 0–1 programming (or pseudo-Boolean optimization). Many polynomially solvable cases have been identified. They include maximization of *almost positive* nonlinear 0–1 functions in which all terms with more than one variable have positive coefficients (Rhys [48], Balinski [1]), *unate* functions which are reducible to almost positive ones by switching some variables (Hansen and Simeone [25], Crama [9], Simeone, de Werra and Cochand [52]), *unimodular* functions which lead to *unimodular* matrices after linearization (Hansen and Simeone [25]), *supermodular* functions (Grötschel, Lovasz and Schrijver [21], Billionnet and Minoux [3]), functions whose co-occurrence graph contains no subgraphs reducible to K_5 (Barahona [2]) and functions whose co-occurrence graph is a *partial k-tree* (Crama, Hansen and Jaumard [10]).

4 Computational results

The algorithm for PSAT, CONDSAT and RSAT described in the previous section has been coded in FORTRAN 77 and extensively tested on a SUN SPARC computer. The resulting program uses the XMP package of Marsten [38] for linear programing. Test problems for PSAT were randomly generated in the following way. The numbers m of sentences, which here are clauses, mc of "conditional" clauses, i.e., constraints associated with conditional probabilities) and n of variables are parameters. Clauses contain at most 4 literals, which is a realistic length (cf, e.g., Buchanan and Shortliffe [4]). There is a uniform distribution of clauses with 1, 2, 3 and 4 literals as well as of complemented variables. Probabilities π_i corresponding to feasible problems were obtained as in Kavvadias and Papadimitriou [31], by generating randomly min(20× number of clauses, 12 000) Boolean vectors x, constructing the corresponding possible worlds, associating with them random numbers summing up to 1 (i.e., probabilities for these worlds to occur), and then summing for each clause the probabilities of the worlds in which it is true. When considering intervals of probability values, a second set of min(20 × number of clauses, 12000) random Boolean vectors was generated. The lower bound of the interval was set up to the smallest probalility obtained, and the upper bound to the largest one.

Results for PSAT with intervals of probability values are presented in Table 1. Ten problems were solved in each case, always until a feasible solution was found. Use of the exact Basic Algorithm Revisited was never required. (This contrasts with results of Kavvadias and Papadimitriou [31], where for many instances of problems with 30 variables or more selecting columns with the Variable Depth Search heuristic did not lead to any feasible solution). Problems with up to 140 variables and up to 300 clauses were solved. Results for the optimization form of PSAT (i.e., Probabilistic Entailment) are summarized in Table 2. Again each line corresponds to averages over 10 problems. Computation times appear to increase less than linearly with the number of variables and, as expected from LP theory, with about the third power of the number of constraints. For small problems the time required by the SAMD heuristic is a large fraction (about 75 % at most) of the total computation times, but for the larger instances this proportion falls to less than 12 % of the total computation time. The exact Basic Algorithm Revisited is called very seldom and takes a very short time. This appears to be due to the fact that when it is called, i.e., when the SAMD heuristic finds no more column with positive reduced cost, most dual variables are equal to 0. The third part of the table corresponds to results obtained when minimization is immediately followed by maximization, taking the optimal basis of the minimization problem as the initial basis for the maximization problem. The computation times are then about 2/3 of the sum of the computation times of the minimization and the maximization problems.

Finally, results for PSAT with probability values, i.e., equality constraints, are given in Table 3. Solution of such PSAT problems appears to take slightly less computing time than of PSAT problems with intervals of probability values, both for the decision and the optimization versions.

Problems with conditional probabilities (CONDSAT) were obtained by modifying some of the first and second series through the addition of constraints involving conditional probabilities of logical sentences. It appears that computation times and other characteristics do not substantially change with this modification when a small number of conditional clauses are considered. Results are given in Tables 4 and 5. We also tested the case with a conditional probability $\pi_{i/j}$ as objective function with an unknown denominator. We observe that few iterations, i.e., updates of the parameter λ are needed. Results are summarized in Table 6.

Finally RSAT problems were obtained by generating clauses as described above and choosing probabilities for these to be true randomly in the interval [0,1] as problems generated in this way are usually infeasible. (Two points are generated following a normal distribution with a mean equal to $0.4 + 0.1\times$ number of variables per clause + alea(0,0.3) where alea(a, b) is a random real number between a and b, and a standard deviation equal to 0.2 / number of literals per clause. The smallest one is the lower bound of the interval and the largest one the upper bound.) Results are presented in Table 7 with averages given over 10 problems. The computing times are about 3 times larger than those of PSAT. This is not surprizing as the number of variables in RSAT is multiplied by three with respect to PSAT. ms is the average number of nonzero dual variables when solving the subproblem with the SAMD heuristic.

		tcpu		pivots		SAMD heuristic		
n	m	mean	σ	mean	σ	iter	mean	σ
50	30	3.46	0.38	116	13	116.6	2.58	0.29
50	40	5.12	1.38	151	42	151.0	3.45	1.06
10	70	9.78	1.70	218	38	218.5	6.20	1.20
20	70	12.63	1.84	265	38	265.3	5.94	0.94
50	70	22.56	5.41	410	117	410.3	10.62	3.21
70	200	808.45	205.71	1955	565	1955.9	100.78	32.04
140	300	11586.53	4806.03	7211	2427	7211.2	737.03	259.58

Table 1: Probabilistic Satisfiability (Decision)

		tcpu		pivots		SAMD heuristic			exact algorithm		
n	m	mean	σ	mean	σ	iter	mean	σ	iter	mean	σ
MAXIMIZATION											
50	30	4.13	0.58	138	16	140.2	3.08	0.39	1.9	0.025	0.008
50	40	6.94	2.91	198	85	200.0	4.64	2.17	1.9	0.031	0.010
10	70	12.83	3.62	290	83	293.0	8.08	2.34	1.7	0.005	0.008
20	70	21.49	7.04	422	130	424.3	9.91	3.49	1.8	0.014	0.008
40	70	28.62	4.88	548	113	550.7	12.94	2.74	2.0	0.029	0.009
50	70	31.24	8.42	553	186	555.2	14.36	5.00	2.0	0.030	0.012
70	70	39.38	8.13	600	144	602.0	19.01	4.67	2.0	0.059	0.016
100	70	51.75	16.22	703	244	705.1	28.94	10.16	2.0	1.080	0.013
140	70	53.05	15.43	576	172	578.9	31.91	9.79	1.9	0.164	0.037
70	100	107.06	23.04	989	235	991.3	34.41	8.60	2.0	0.053	0.012
70	150	520.55	202.07	2208	984	2210.2	95.58	46.98	1.8	0.051	0.018
70	200	1222.90	290.98	2663	751	2665.6	137.14	40.79	2.0	0.061	0.022
MINIMIZATION											
50	30	3.37	0.61	114	19	116.7	2.53	0.46	2.0	0.027	0.007
50	40	5.80	1.53	156	44	158.1	3.95	1.20	2.0	0.034	0.013
10	70	12.83	3.75	285	80	287.4	7.95	2.33	2.0	0.010	0.010
20	70	13.61	2.62	294	59	296.1	6.42	1.36	2.0	0.010	0.010
40	70	21.74	3.08	419	71	421.4	9.89	1.76	2.0	0.022	0.008
50	70	25.11	6.75	446	148	448.1	11.68	3.94	2.0	0.028	0.010
70	70	28.69	5.25	441	93	443.7	14.01	3.12	2.0	0.055	0.011
100	70	38.54	7.81	523	128	525.8	21.49	5.32	2.0	0.096	0.016
140	70	45.51	8.15	501	96	503.8	27.61	5.62	2.0	0.171	0.022
70	100	84.22	16.97	785	182	787.1	27.32	6.60	2.0	0.054	0.013
70	150	393.05	145.83	1734	714	1736.5	74.63	33.50	1.9	0.055	0.008
70	200	1014.02	295.84	2322	733	2324.5	118.73	40.81	2.0	0.051	0.012
MINIMIZATION + MAXIMIZATION											
50	30	5.88	0.84	191	27	195.4	4.32	0.59	4.1	0.065	0.023
50	40	8.58	2.61	242	85	246.0	5.63	2.02	4.2	0.060	0.018
10	70	18.41	6.45	407	146	411.4	11.23	4.12	3.5	0.014	0.011
40	70	35.84	5.65	670	132	674.8	15.83	3.21	4.6	0.053	0.018
70	70	48.60	9.05	728	158	732.2	23.10	5.21	4.1	0.110	0.021
100	70	65.50	14.37	879	239	883.4	36.13	9.91	4.3	0.218	0.036
140	70	71.06	16.26	737	193	761.1	42.39	11.05	4.0	0.347	0.069
70	100	141.62	20.57	1263	228	1267.8	43.91	8.30	4.0	0.101	0.020
70	150	601.63	228.20	2525	1117	2529.8	108.38	51.69	3.8	0.106	0.032
70	200	1628.40	417.68	3345	1018	3349.2	177.57	58.40	4.0	0.117	0.040

Table 2: Probabilistic Satisfiability (Optimization)

		tcpu		pivots		SAMD heuristic				exact algorithm				
							tcpu		terms		tcpu		terms	variables
n	m	mean	σ	mean	σ	iter	mean	σ	mean	iter	mean	σ	mean	mean
DECISION														
50	70	18.00	3.82	325	79	325.9	8.56	2.22	70.93	—	—	—	—	—
70	70	23.86	5.64	372	101	372.0	12.17	3.50	70.94	—	—	—	—	—
100	70	31.60	4.56	417	56	417.2	17.55	2.61	70.89	—	—	—	—	—
140	70	40.00	7.38	437	91	437.9	24.91	5.03	70.90	—	—	—	—	—
70	100	65.50	14.41	629	164	629.7	23.10	6.47	100.98	—	—	—	—	—
70	150	233.24	37.49	1110	173	1110.9	49.31	8.43	150.99	—	—	—	—	—
70	200	778.91	253.43	1960	643	1960.5	102.89	35.65	200.95	0.4	0.160	0.034	8.001	8.00
140	300	10699.55	7663.09	7061	4319	7061.9	726.09	473.03	300.79	—	—	—	—	—
OPTIMIZATION (MINIMIZATION)														
50	70	19.49	4.68	346	93	348.8	9.17	2.60	70.52	2.0	.029	.012	1.30	2.50
70	70	26.40	6.61	407	106	409.8	13.29	3.70	70.52	2.1	.055	.011	1.52	2.52
100	70	34.20	6.20	436	73	438.9	18.97	3.38	70.64	2.0	.098	.017	1.30	2.90
140	70	42.35	11.73	464	132	466.9	26.09	7.75	70.56	2.0	.183	.027	1.10	3.00
70	100	66.16	11.48	630	138	632.5	23.25	5.50	100.64	2.0	.052	.015	1.10	3.10
70	150	265.49	62.12	1224	271	1226.3	54.03	13.14	150.72	2.0	.050	.016	2.00	2.80
70	200	870.67	291.20	2131	662	2133.1	112.94	38.09	200.80	2.0	.054	.016	2.90	3.90

Table 3: Probabilistic Satisfiability with equality constraints

			tcpu		pivots		SAMD heuristic		
n	m	mc	mean	σ	mean	σ	iter	mean	σ
50	30	3	3.50	0.53	116	16	116.5	2.52	0.40
50	40	4	6.55	1.10	185	31	185.0	44.01	0.80
10	70	7	12.99	2.73	262	60	262.2	7.93	1.82
20	70	7	13.86	1.43	286	27	286.9	6.58	0.80
50	70	7	24.17	5.46	410	104	410.0	10.83	2.91
70	200	20	869.55	287.52	2333	882	2333.3	124.76	51.37

Table 4: Conditional Probabilistic Satisfiability (Decision)

			tcpu		pivots		SAMD heuristic			exact algorithm		
n	m	mc	mean	σ	mean	σ	iter	mean	σ	iter	mean	σ
MAXIMIZATION												
50	30	3	4.41	1.0	142	32	144.6	3.12	0.75	1.9	0.025	0.008
50	40	4	8.56	1.81	234	47	236.7	5.42	1.14	1.9	0.036	0.029
10	70	7	19.76	2.52	394	40	396.1	11.60	1.44	1.7	0.144	0.431
20	70	7	23.29	11.19	409	108	411.9	9.34	2.40	2.0	2.824	8.888
50	70	7	29.36	6.87	489	121	419.8	12.96	3.42	1.6	0.025	0.015
70	200	20	1203.40	475.64	3022	1256	302.4	162.02	73.34	1.9	0.051	0.017
MINIMIZATION												
50	30	3	3.50	0.55	114	16	116.9	2.50	0.37	2.0	0.030	0.011
50	40	4	6.78	0.92	187	27	189.5	4.34	0.66	2.0	0.032	0.010
10	70	7	15.85	3.96	324	78	326.5	9.56	2.30	1.9	0.008	0.008
20	70	7	14.91	1.12	298	28	300.7	6.97	0.88	2.0	0.008	0.008
50	70	7	26.68	5.95	441	117	443.7	11.66	3.25	2.0	0.029	0.009
70	200	20	1235.03	444.58	2693	1023	269.6	145.00	59.62	2.0	0.052	0.009

Table 5: Conditional Probabilistic Satisfiability (Optimization)

			tcpu		pivots		updates	SAMD heuristic				exact algorithm				
									tcpu		terms		tcpu		terms variables	
n	m	mc	mean	σ	mean	σ	λ	iter	mean	σ	mean	iter	mean	σ	mean	mean
70	70	0	41.74	8.17	610	135	4.8	620.7	20.06	4.46	69.3	10.0	0.263	0.091	1.62	3.42
70	100	0	116.37	33.11	1044	341	5.2	1054.7	38.18	13.27	99.5	10.5	0.300	0.071	1.40	4.50
70	70	20	78.85	21.19	820	216	4.4	829.0	28.81	7.84	88.6	8.7	0.254	0.127	1.57	4.55

Table 6: Conditional Probabilistic Satisfiability with hyperbolic objective functions (Optimization)

		tcpu		pivots		SAMD heuristic				exact algorithm				
							tcpu		terms		tcpu		terms variables	
n	m	mean	σ	mean	σ	iter	mean	σ	mean	iter	mean	σ	mean	mean
10	70	20.17	4.21	497	66	499.7	13.80	2.18	61.6	2.5	1.051	1.496	34.3	9.76
40	70	52.64	8.86	1183	215	1185.8	28.02	5.33	68.4	2.0	0.025	0.010	4.9	4.70
70	70	70.93	14.95	1242	269	1244.3	40.47	8.55	68.9	2.0	0.055	0.014	3.4	2.70
70	100	144.89	33.43	4035	951	4037.9	144.89	33.43	98.9	2.0	0.061	0.018	6.4	6.00

Table 7: Restore Probabilistic Satisfiability

5 Conclusions

Nilsson [42] has presented a basic model for rigorous reasoning under uncertainty, which is compatible with classical probability theory. He showed that the PSAT problem could be expressed as a linear program. Georgakopoulos, Kavvadias and Papadimitriou [14] [31] proposed to solve PSAT by column generation, using a Varaible Depth Search heuristic in the selection of the entering column. Continuing the investigation of this approach, with recent tools from nonlinear 0–1 programming, i.e., the SAMD heuristic and the Basic Algorithm Revisited, we propose an exact algorithm for solving large instances of PSAT (in decision or optimization form, and possibly with interval constraints for truth probabilities of sentences). A computer program for this algorithm allows to solve readily problems with up to 140 variables and 300 constraints. Moreover, we extend Nilsson's model and the algorithm to address constraints on conditional probabilities (the CONDSAT problem), conditional probabilities in the objective function and to restore consistency of inconsistent problems with minimal changes in the ranges of the probabilities (RSAT problem).

Acknowledgements: Work of the first and third authors has been supported by NSERC grant # GP0036426 and FCAR grants # 89EQ4144 and # 90NC0305. Work of the second author was supported by AFORS grant 0066 to Rutgers University.

The authors are grateful to Jacques Desrosiers for several fruitful discussions on column generation methods and to Thierry Brun and Olivier Bahn for their help in the programming part of the early versions of the algorithm.

References

[1] M. BALINSKI, 1970. *On a selection problem*, **Management Science 17**, 230–231.

[2] F. BARAHONA, 1983. *The max-cut problem on graphs not contractible to K_5*, **Operations Research Letters 2:3**, 107–111.

[3] A. BILLIONNET and M. MINOUX, 1985. *Maximizing a supermodular pseudo-Boolean function: a polynomial algorithm for supermodular cubic functions*, **Discrete Applied Mathematics 12**, 1–11.

[4] B.G. BUCHANAN and E.H. SHORTLIFFE, 1985. *Rule-based expert systems — The* MYCIN *experiments of the Stanford heuristic programming project*, Addison-Wesley, Reading, MA.

[5] A. BUNDY, 1985. *Incidence calculus: a mechanism for probabilistic reasoning*, **Journal of Automated Reasoning 1**, 263–283.

[6] C.L. CHANG and R.C.T. LEE, 1973. *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York.

[7] S.S. CHEN, *Some extensions of probabilistic logic*, in [35], 221–227.

[8] V. CHVATAL, 1980. *Linear Programming*, Freeman, New York.

[9] Y. CRAMA, 1989. *Recognition problems for special classes of polynomials in 0–1 variables*, **Mathematical Programming 44**, 139–155.

[10] Y. CRAMA, P. HANSEN and B. JAUMARD, *The basic algorithm for minimizing nonlinear pseudo-Boolean functions revisited*, to appear in **Discrete Applied Mathematics**.

[11] D. DUBOIS and H. PRADE, 1987. *A tentative comparison of numerical approximate reasoning methodologies*, **International Journal Man-Machine Studies 27**, 717–728.

[12] D. DUBOIS and H. PRADE, 1987. *Necessity measures and the Resolution Principle*, **IEEE Transactions on Systems, Man and Cybernetics SMC-17:3**, 474–478.

[13] M.R. GENESERETH and N.J. NILSSON, 1987. *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA.

[14] G. GEORGAKOPOULOS, D. KAVVADIAS and C.H. PAPADIMITRIOU, 1988. *Probabilistic satisfiability*, **Journal of Complexity 4**, 1–11.

[15] P.C. GILMORE and R.E. GOMORY, 1961. *A linear programming approach to the cutting stock problem*, **Operations Research 9**, 849–859.

[16] F. GLOVER, 1986. *Future paths for integer programming and links for Artificial Intelligence*, **Computers and Operations Research 13:5**, 533–549.

[17] F. GLOVER, 1989. *Tabu search - Part I*, **ORSA Journal on Computing 1:3**, 190–206.

[18] J. GORDON and E.H. SHORTLIFFE, 1985. *A method for managing evidential reasoning in a hierarchical hypothesis space*, **Artificial Intelligence 26**, 323–357.

[19] B.N. GROSOF, *An inequality paradigm for probabilistic reasoning*, in [34], 259–275.

[20] B.N. GROSOF, *Non-monotonicity in probabilistic knowledge*, in [35], 237–250.

[21] M. GRÖTSCHEL, L. LOVASZ and A. SCHRIJVER, 1981. *The ellipsoid method and its consequences in combinatorial optimization*, **Combinatorica 1**, 169–197. (Corrigendum 4 (1984) 291–295).

[22] P.L. HAMMER, I. ROSENBERG and S. RUDEANU, 1963. *On the determination of the minima of pseudo-boolean functions*, (in Romanian), **Studii si Cercetari matematice 14**, 359–364.

[23] P.L. HAMMER and S. RUDEANU, 1968. *Boolean methods in Operations Research and Related Areas*, Springer-Verlag, Berlin.

[24] P. HANSEN and B. JAUMARD, *Algorithms for the maximum satisfiability problem*, to appear in **Computing**.

[25] P. HANSEN and B. SIMEONE, 1986. *Unimodular functions*, **Discrete Applied Mathematics 14**, 269–281.

[26] M. HENRION, 1987. *Uncertainty in artificial intelligence: is probability epistemologically and heuristically adequate?*, in J.L. Mumpower et al. (eds.), **Expert judgement and expert systems**, NATO ASI series F: Computer and systems sciences 35, 105–129.

[27] M. HENRION, *Propagating uncertainty in Bayesian networks by probabilistic logic sampling*, in [35], 149–164.

[28] E.J. HORVITZ, D.E. HECKERMAN, and C.P. LANGLOTZ, 1986. *A framework for comparing formalisms for plausible reasoning*, **Proceedings of the AAAI**, Morgan Kaufman, San Mateo, CA.

[29] J.R. ISBELL and W.H. MARLOW, 1956. *Attribution games*, **Naval Research Logistics Quaterly 3**, 71–94.

[30] B. JAUMARD, 1986. *Extraction et utilisation des relations booléennes pour la résolution des programmes l inéaires en variables 0–1*, Thèse de Doctorat, École Nationale Supérieure des Télécommunications, Paris.

[31] D. KAVVADIAS and C.H. PAPADIMITRIOU, *A linear programming approach to reasoning about probabilities*, to appear in **Annals of Mathematics and Artificial Intelligence.**

[32] L.G. KHACHIYAN, 1979. *A polynomial algorithm in linear programming* (in Russian), **Doklady Akademii Nauk SSSR 244**, 1979, 1093–1096. (English translation: **Soviet Mathematics Doklady 20**, 1979, 191–194).

[33] L.S. LASDON, 1970. *Optimization Theory for Large Systems*, Macmillan, New York.

[34] J.F. LEMMER and L.N. KANAL (eds.), 1986. *Uncertainty in Artificial Intelligence* 1, North-Holland, Amsterdam.

[35] J.F. LEMMER and L.N. KANAL (eds.), 1988. *Uncertainty in Artificial Intelligence* 2, North-Holland, Amsterdam.

[36] S. LIN and B.W. KERNIGHAN, 1970. *An efficient heuristic procedure for partitioning graphs*, **Bell Systems Technical Journal 49:2**, 291-307.

[37] S. LIN and B.W. KERNIGHAN, 1973. *An effective heuristic for the traveling salesman problem*, **Operations Research 21**, 498-516.

[38] R.E. MARSTEN, 1981. *The Design of the XMP linear programming library*, **ACM Tansactions on Mathematical Software 7:4**, 481–497.

[39] M. MCLEISH, *Probabilistic logic: some comments and possible use for nonmonotonic reasoning*, in [35], 55–62.

[40] K.S. LEUNG and W. LAM, 1988. *Fuzzy concepts in expert systems*, **Computer**, 43–56.

[41] W.E. NELSON, 1982. REACTOR: *an expert system for diagnosis and treatment of nuclear reactor accidents*, **Proceedings of the National Conference of Artificial Intelligence.**

[42] N.J. NILSSON, 1986. *Probabilistic logic*, **Artificial Intelligence 28:1**, 71–87.

[43] G. PAASS, 1988. *Probabilistic logic*, in P. Smets et al. (eds.) **Non-standard Logics for Automated Reasoning**, Academic Press, New York, 213–251.

[44] J. PEARL, 1986. *Fusion, propagation and structuring in belief networks*, **Artificial Intelligence 29**, 241–288.

[45] J. PEARL, 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Morgan Kaufman, San Mateo, CA.

[46] K. KONOLIGE, 1979. *An inference net compiler for the* PROSPECTOR *rule-based consultation systems*, **Proceedings of the International Joint Conference on Artificial Intelligence 6**, 487–489.

[47] R. REITER, 1987. *Nonmonotonic reasoning*, **Annual Reviews on Computer Science 2**, 147–186.

[48] J.M.W. RHYS, 1970. *A selection problem of shared fixed costs and network flows*, **Management Science 17**, 200–207.

[49] S. SCHAIBLE and T. IBARAKI, 1983. *Fractional programming*, **European Journal of Operational Research 12**, 325–338.

[50] G. SHAFER, 1976. *A mathematical theory of evidence*, Princeton University, Princeton, NJ.

[51] S.K. SHUM, J.F. DAVIS, W.F. PUNCH and B. CHANDRASEKARAN, 1988. *An expert system approach to malfunction diagnosis in chemical plants*, **Comput. Chem. Engineering 12:1**, 27–36.

[52] B. SIMEONE, D. DE WERRA and M. COCHAND, *Combinatorial properties and recognition of some classes of unimodular functions*, to appear in **Discrete Applied Mathematics.**

[53] H.S. STEPHANOU and A.P. SAGE, 1987. *Perspectives on imperfect information processing*, **IEEE Transactions on Systems, Man and Cybernetics 17**, 780–798.

[54] P. SUPPES, 1966. *Probabilistic Inference and the Concept of Total Evidence*, in Hintikka, J. and Suppes, P. (eds.), **Aspects of Inductive Logic**, North-Holland, Amsterdam, 49–65.

[55] L.A. ZADEH, 1971. *Fuzzy sets*, **Information and Control 8**, 338–353.

Computational experience with an interior point algorithm on the Satisfiability problem

A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan and M.G.C. Resende

Mathematical Sciences Research Center
AT&T Bell Laboratories, Murray Hill, NJ 07974 USA

Abstract

We apply the zero-one integer programming algorithm described in Karmarkar [12] and Karmarkar, Resende and Ramakrishnan [13] to solve randomly generated instances of the satisfiability problem (SAT). The interior point algorithm is briefly reviewed and shown to be easily adapted to solve large instances of SAT. Hundreds of instances of SAT (having from 100 to 1000 variables and 100 to 32,000 clauses) are randomly generated and solved. For comparison, we attempt to solve the problems via linear programming relaxation with MINOS.

Key words: Integer programming, interior point method, logic, satisfiability.

1. Introduction

We consider here the satisfiability problem (SAT) in propositional calculus, a central problem in mathematical logic that was posed as the original NP-complete problem [3].

A boolean *variable* x is a variable that can assume only the values `true` or `false`. Boolean variables can be combined by the logical connectives or ($\vee$), and ($\wedge$) and not ($\bar{x}$) to form boolean formulae. A variable or a single negation of the variable is called a *literal.* A boolean formula consisting of only literals combined by just the $\vee$ connector is called a *clause.*

The Satisfiability Problem (SAT) can be defined as follows. Given n clauses $C_1, \ldots, C_n$ involving m variables $x_1, \ldots, x_m$, is the formula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n \tag{1}$$

satisfiable? What is wanted is an assignment of truth values to the boolean variables so that the boolean formula (1) has value `true`. Hence, the assignment of truth values to the boolean variables must make each clause $C_1, C_2, \ldots, C_n$ `true`. We present a method for finding a satisfiable truth assignment, based on the interior point zero-one integer programming algorithm described in Karmarkar [12] and Karmarkar, Resende and Ramakrishnan [13]. Once we have a truth assignment, it is trivial to prove satisfiability by substitution. In case the formula is not satisfied, our method does not construct a proof of nonsatisfiability.

Consider the following three clauses,

$$\begin{aligned} C_1 &= x_1 \vee \bar{x}_2 \vee x_3 \\ C_2 &= x_2 \vee x_3 \\ C_3 &= x_1 \vee \bar{x}_3. \end{aligned}$$

The formula $C_1 \wedge C_2 \wedge C_3$ is clearly satisfiable, since the truth assignment $x_1 =$ `true`, $x_2 =$ `true` and $x_3 =$ `false` makes each clause C_1, C_2 and C_3 `true`.

There has been great interest in devising efficient algorithms for solving the satisfiability problem. One obvious way would be to try all possible truth assignments to see if one satisfies the formula. This has exponential complexity, since in the worst case one may have to try 2^m truth assignments. The Davis-Putnam procedure [4] is a technique that has been shown to take polynomial average time complexity, subject to certain restrictions. Unfortunately, these constraints limit substantially the application of the algorithm. Resolution is a widely used method for inference problems in propositional calculus [19]. Unfortunately, resolution not only has exponential worst case complexity but fails to solve even moderately sized inference problems. The resolution method can find unsatisfiability quickly in certain classes of problems but is slow when it comes to verifying satisfiability, as all possible resolutions need to be tried out before concluding that the inference relation holds or that the problem is satisfiable. Hooker [9] has shown that mathematical programming methods are substantially faster than resolution.

An interesting aspect of the problem is that it can be formulated as an integer program. The integer programming formulation is immediate if one identifies logical value `true` with the integer 1 and `false` with -1. The following procedure is standard to transform a clause into a linear inequality. First, all $\vee$ connectives are transformed into the $+$ of ordinary addition. The literal x is represented by the integer variable w and $\bar{x}$ is transformed into a $-w$. We impose the condition that each clause is `true`. Hence, if a clause C has $n(C)$ literals then at least one must be `true`, i.e. contribute to a $+1$, while the rest are free to be `true` or `false`, thus contributing at least -1 each, resulting in the constraint

$$\sum_{j \in I_C} w_j - \sum_{j \in J_C} w_j \geq 2 - n(C),$$

where I_C is the set of indices of all $+w$ variables in clause C, J_C is the set of indices of all $-w$ variables in C and $n(C) = |I_C| + |J_C|$. Hence, the clauses in the earlier example would give us the constraints

$$\begin{aligned} w_1 - w_2 + w_3 &\geq -1 \\ w_2 + w_3 &\geq 0 \\ w_1 - w_3 &\geq 0 \end{aligned}$$

with the restriction that $w_j = \pm 1, \ j = 1, 2, 3$, requiring that extra constraints be added to insure that each w_j be in the closed interval $[-1, 1]$. These constraints are

$$-1 \leq w_1 \leq 1$$

$$-1 \leq w_2 \leq 1$$

$$-1 \leq w_3 \leq 1.$$

This ± 1 integer programming formulation of SAT is similar to the more common form of integer programming, where variables x_j take on $(0,1)$ values. The ± 1 problem can be transformed into the $(0,1)$ form with the change of variables

$$x_j = \frac{1+w_j}{2}, \quad j = 1, \ldots, m.$$

Several authors have investigated integer programming approaches to resolving boolean logic problems, e.g. [1], [5], [6], [7], [8], [9], [10], [11], [22], [21], [23]. A recent survey is given in [2].

Recently a cutting plane algorithm that uses resolvents as cuts has been introduced by Hooker [9], who has reported good computational results for certain problem classes. However, the method, based on the simplex method of linear programming, has been described ineffective on dense satisfiability problems. Moreover, as will be discussed later, Hooker allows for the presence of unit clauses in his sample problems which contribute toward making the problems less difficult than instances of SAT in which single literal clauses are disallowed. Hooker reports computational results for small sized problems and the behavior of his method for larger instances is hard to predict.

An outline of the remainder of this paper is as follows. In section 2 we briefly review the integer programming algorithm. Specific details required to apply the integer programming algorithm to SAT are given in section 3. Computational results are provided in section 4. In section 5 we make concluding remarks.

2. Zero-One Integer Programming

In this section, we briefly review the interior point algorithm used in this study. We omit all proofs. They can be found in [12] and [14]. We consider the following integer programming problem:
INTEGER PROGRAMMING: Let $B \in \Re^{m \times n}$ and $b \in \Re^n$. Find $w \in \Re^m$ such that:

$$B^T w \leq b \tag{2}$$

$$w_i = \pm 1, \; i = 1, \ldots, m \tag{3}$$

We propose an interior point approach to solve (2-3), i.e. a heuristic that generates a sequence of points $\{w^0, w^1, \ldots, w^k, \ldots\}$ where for all $k = 0, 1, \ldots$

$$w^k \in \left\{ w \in \Re^m | B^T w < b; \; -e < w < e \right\},$$

where $e^T = (1, \ldots, 1)$. In practice, this sequence often converges to a point from which one can round off to a ± 1 integer solution to (2-3). No guarantee can be made as to whether the heuristic will be successful, but in this paper we provide several large instances of SAT where it succeeds in proving satisfiability.

To simplify notation, let I denote an $m \times m$ identity matrix,

$$A = \left[B \vdots I \vdots -I \right]$$

and

$$c = \begin{bmatrix} b \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

and let

$$\mathcal{I} = \{w \in \Re^m \mid A^T w \leq c \text{ and } w_i = \pm 1\}.$$

With this notation, INTEGER PROGRAMMING can be restated as: Find $w \in \mathcal{I}$.

Throughout this paper we assume that A^T has full rank. Let

$$\mathcal{L} = \{w \in \Re^m \mid A^T w \leq c\}$$

and consider the linear programming relaxation of (2-3), i.e. find $w \in \mathcal{L}$. One way of selecting ± 1 integer solutions over fractional solutions in linear programming is to introduce the quadratic objective function,

$$\text{maximize } w^T w = \sum_{i=1}^{m} w_i^2$$

and solve the NP-complete [20] nonconvex quadratic programming problem

$$\text{maximize } w^T w \tag{4}$$

$$\text{subject to : } A^T w \leq c \tag{5}$$

The following proposition establishes the relationship between (4-5) and INTEGER PROGRAMMING.

Proposition 2.1 *Let $w \in \mathcal{L}$. Then $w \in \mathcal{I} \Longleftrightarrow w^T w = m$, where m is the optimal solution to (4-5).*

We now consider an algorithm to solve (4-5) and show how to apply this algorithm to integer programming. Let

$$w^0 \in \mathcal{L}_s = \{w \in \Re^m \mid A^T w < c\}$$

be a given initial interior point. The algorithm generates a sequence of interior points of $\mathcal{L}$. Let $w^k \in \mathcal{L}_s$ be the k-th iterate. Around w^k we construct a quadratic approximation of the potential function

$$\varphi(w) = \log \sqrt{m - w^T w} - \frac{1}{n} \sum_{k=1}^{n} \log d_k(w)$$

where

$$d_k(w) = c_k - a_k^T w, \quad k = 1, \ldots, n.$$

Let $D = \text{diag}(d_1(w), \ldots, d_n(w))$, $e = (1, \ldots, 1)$, $f_0 = m - w^T w$ and C be a constant. The quadratic approximation of $\varphi(w)$ around w^k is given by

$$Q(w) = \frac{1}{2}(w - w^k)^T H (w - w^k) + h^T (w - w^k) + C \tag{6}$$

where the Hessian is

$$H = -\frac{2}{f_0} I - \frac{4}{f_0^2} w^k w^{kT} + \frac{1}{n} A D^{-2} A^T \tag{7}$$

and the gradient is

$$h = -\frac{1}{f_0} w^k + \frac{1}{n} A D^{-1} e. \tag{8}$$

Minimizing (6) subject to $A^T w \leq c$ is NP-complete. However, if the polytope is approximated by an inscribed ellipsoid,

$$\mathcal{E}(r) = \left\{ w \in \Re^m | (w - w^k)^T A D^{-2} A^T (w - w^k) \leq r^2 \leq 1 \right\}, \tag{9}$$

the resulting approximate problem,

$$\text{minimize } \frac{1}{2} (\Delta w)^T H \Delta w + h^T \Delta w \tag{10}$$

$$\text{subject to : } (\Delta w)^T A D^{-2} A^T (\Delta w) \leq r^2 \leq 1, \tag{11}$$

where $\Delta w \equiv w - w^k$, is easy.

We now consider an algorithm to solve INTEGER PROGRAMMING based on nonconvex quadratic programming. The approach used to solve the nonconvex optimization problem

$$\text{minimize } \{\varphi(w) | A^T w \leq c\},$$

is similar to the classical Levenberg-Marquardt methods, [16], [17], first suggested in the context of nonlinear least squares. This algorithm solves (10-11) to produce a descent direction Δw^* for the potential function $\varphi(w)$. A solution $\Delta w^* \in \Re^m$ to (10-11) is optimal if and only if there exists $\mu \geq 0$ such that:

$$\Delta w^* \left(H + \mu A D^{-2} A^T \right) = -h \tag{12}$$

$$\mu \left((\Delta w^*)^T A D^{-2} A^T \Delta w^* - r^2 \right) = 0 \tag{13}$$

$$H + \mu A D^{-2} A^T \text{ is positive semi-definite.} \tag{14}$$

With the change of variables $\gamma = 1/(\mu + 1/n)$ and substituting (7) and (8) into (12) we obtain an expression for Δw^* satisfying (12):

$$\Delta w^* = -\left(A D^{-2} A^T - \frac{4\gamma}{f_0^2} w^k w^{k^T} - \frac{2\gamma}{f_0} I \right)^{-1} \gamma \left(-\frac{1}{f_0} w^k + \frac{1}{n} A D^{-1} e \right) \tag{15}$$

Note that r does not appear in (15). However, (15) is not defined for all values of r. It can be shown that if the radius r of the ellipsoid (11) is kept within a certain size, then there exists an interval $0 \leq \gamma \leq \gamma_{max}$ such that

$$A D^{-2} A^T - \frac{4\gamma}{f_0^2} w^k w^{k^T} - \frac{2\gamma}{f_0} I \tag{16}$$

is nonsingular. The following proposition establishes a descent direction of $\varphi(w)$.

Proposition 2.4 *There exists $\gamma > 0$ such that the direction Δw^*, given in (15), is a descent direction of $\varphi(w)$.*

We now outline the steps of the algorithm to find a solution of (10-11) satisfying condition (12-14) and show how to incorporate this approach into an algorithm to solve INTEGER PROGRAMMING. Each iteration of this algorithm is comprised of two tasks. To simplify notation, let

$$H_c = AD^{-2}A^T \tag{17}$$

$$H_o = -\frac{4}{f_0^2}w^k w^{kT} - \frac{2}{f_0}I \tag{18}$$

and define

$$M = H_c + \gamma H_o.$$

Given the current iterate w^k we first seek a value of γ such that $M\Delta w = \gamma h$ has a solution Δw^*. This can be done by binary search, as we will see shortly. Once such a parameter γ is found, the linear system

$$M\Delta w^* = \gamma h \tag{19}$$

is solved for $\Delta w^* \equiv \Delta w^*(\gamma(r))$. It is easy to verify that the *length*,

$$l(\Delta w^*(\gamma)) = (\Delta w^*(\gamma(r)))^T AD^{-2}A^T \Delta w^*(\gamma(r),$$

is a monotonically increasing function of γ in the interval $0 \leq \gamma \leq \gamma_{max}$.

Optimality condition (13) implies that $r = \sqrt{l(\Delta w^*(\gamma))}$ if $\mu > 0$. Small lengths result in small changes in the potential function, since r is small and the optimal solution lies on the surface of the ellipsoid. A length that is too large may not correspond to an optimal solution of (10-11), since this may require $r > 1$. We maintain an interval $(\underline{l}, \bar{l})$ called the *acceptable length region* and accept a length $l(\Delta w^*(\gamma))$ if $\underline{l} \leq l(\Delta w^*(\gamma)) \leq \bar{l}$. If $l(\Delta w^*(\gamma)) < \underline{l}$, γ is increased and (19) is resolved with the new M matrix and h vector. On the other hand, if $l(\Delta w^*(\gamma)) > \bar{l}$, γ is reduced and (19) is resolved. We shall say a *local minimum* has been found if the length is reduced below a tolerance ϵ. If a local minimum is found, several strategies can be considered. In one approach, the problem is modified (by adding a cut, for example) and the algorithm is applied to the new problem. Once an acceptable length is produced the new iterate w^{k+1} is computed by moving in direction $\Delta w^*(\gamma)$ from w^k with a step size $\alpha < 1$,

$$w^{k+1} = w^k + \alpha \Delta w^*(\gamma). \tag{20}$$

The current iterate w^{k+1} is rounded off to the nearest ± 1 vertex: $\tilde{w}^{k+1} = (\pm 1, \ldots, \pm 1)$. If $\tilde{w}^{k+1}$ is such that $A^T \tilde{w}^{k+1} \leq c$ then $\tilde{w}^{k+1}$ is a global optimal solution of (4-5).

Pseudo code 2.1 details procedure `ip`, the integer programming algorithm that makes use of procedure `descent_direction` to optimize (10-11) producing the descent direction. In `ip`, procedure `get_start_point` returns an initial starting interior point and procedure `round_off` rounds a fractional solution to a ± 1 integer solution. These procedures are discussed in more detail in section 3.

Pseudo code 2.2 details procedure `descent_direction`.

```
procedure ip(A, c, γ0, l̲0, l̄0)
1    k := 0;  γ := γ0;  l̲ := l̲0;  l̄ := l̄0;  K := 0;
2    w^k := get_start_point(A, c);
3    w̃^k := round_off(w^k);
4    do A^T w̃^k ≱ c →
5        Δw* := descent_direction(γ, w^k, l̲, l̄);
6        do φ(w^k + αΔw*) ≥ φ(w^k)  and  l̄ > ε →
7            l̄ := l̄/l̄_r;
8            Δw* := descent_direction(γ, w^k, l̲, l̄)
9        od;
10       if φ(w^k + αΔw*) < φ(w^k) →
11           w^{k+1} := w^k + αΔw*;
12           w̃^{k+1} := round_off(w^{k+1});
13           k := k + 1
14       fi;
15       if l̄ ≤ ε →
16           print("Converged to local minimum.");
17           exit()
18       fi
19   od
end ip;
```

Pseudo-Code 2.1 - The ip Algorithm

```
procedure descent_direction(γ, w^k, l̲, l̄)
1    l := ∞;  LD_key := false;  γ̄_key := false;  γ̲_key := false;
2    do l > l̄  or  (l < l̲  and  LD_key = false) →
3        M := H_c + γH_o;  b := γh;
4        do MΔw = b has no solution →
5            γ := γ/γ_r;  LD_key := true;
6            M := H_c + γH_o;  b := γh
7        od;
8        Δw* := M^{-1}b;  l := (Δw*)^T AD^{-2}A^T Δw*;
9        if l < l̲  and  LD_key = false →
10           γ̲ := γ;  γ̲_key := true;
11           if γ̄_key = true → γ := √(γ̲γ̄) fi;
12           if γ̄_key = false → γ := γ · γ_r fi
13       fi;
14       if l > l̄ →
15           γ̄ := γ;  γ̄_key := true;
16           if γ̲_key = true → γ := √(γ̲γ̄) fi;
17           if γ̲_key = false → γ := γ/γ_r fi
18       fi
19   od;
20   do l < l̲  and  LD_key = true → l̲ := l̲/l_r od;
21   return(Δw*)
end descent_direction;
```

Pseudo-Code 2.2 - The `descent_direction` Algorithm

3. Application Specific Details

In this section, we discuss algorithmic details specific to this application (SAT).

3.1. Initial Solution

The algorithm requires an initial interior point solution to

$$\sum_{j \in Ic_i} w_j - \sum_{j \in Jc_i} w_j \geq 2 - n(C_i), \quad i = 1, \ldots, n \tag{21}$$

$$-1 \leq w_j \leq 1, \quad j = 1, \ldots, m. \tag{22}$$

One way to obtain such a starting solution is to solve a Phase I artificial linear program and use its solution as the initial solution for the integer programming algorithm. Instead, we simply use the origin $w^0 = (0, \ldots, 0)$ as the initial solution because, in practice, there is no apparent benefit from the linear programming approach. However, the origin is not an interior point to (21-22) if there exists any clause C_i such that $n(C_i) = 2$. To get around this problem, we use an alternate right hand side to (21), resulting in

$$\sum_{j \in Ic_i} w_j - \sum_{j \in Jc_i} w_j \geq 1 - n(C_i), \quad i = 1, \ldots, n \tag{23}$$

$$-1 \leq w_j \leq 1, \quad j = 1, \ldots, m. \tag{24}$$

Proposition 3.1 *If $n(C_i) \geq 2$, $i = 1, \ldots, n$ and $w_j = \pm 1$, $j = 1, \ldots, m$, then (23-24) is a valid description of SAT.*

Proof: We first show that $w^0 = (0, \ldots, 0)$ satisfies the constraints. Constraints (24) are clearly satisfied. Constraints (23) are also satisfied, because

$$\sum_{j \in Ic_i} w_j^0 - \sum_{j \in Jc_i} w_j^0 = 0 > 1 - n(C_i), \quad i = 1, \ldots, n,$$

since $n(C_i) \geq 2$, $i = 1, \ldots, n$. We must now show that for all ± 1 integer solutions, at least one $w_j = 1$ for $j \in Ic_i$ or at least one $w_j = -1$ for $j \in Jc_i$ for $i = 1, \ldots, n$. Assume the contrary, i.e. assume $\forall j \in Ic_i$, $w_j = -1$ and $\forall j \in Jc_i$, $w_j = 1$. Then $-|Ic_i| - |Jc_i| \geq 1 - |Ic_i| - |Jc_i|$, which implies $0 \geq 1$, a contradiction. □

3.2. Rounding Off

Rounding off can be done in a straightforward manner, for $j = 1, \ldots, m$,

$$\tilde{w}_j = \begin{cases} +1 & \text{if } w_j > 0 \\ -1 & \text{if } w_j \leq 0. \end{cases}$$

We call this rounding scheme *type A*.

Rounding scheme A does not discriminate between, say, positive interior point variables $w_{k_0} = 0.01$ and $w_{k_1} = 0.99$, assigning both variables a $+1$. Intuitively however, one should expect the boolean variable x_{k_1} to be `true` with higher probability than variable x_{k_0}. Furthermore, scheme A takes no advantage of the structure of the constraints. Assigning a truth value to a boolean variable may force some other variable to take on a certain truth value. For example, if clause $C : \bar{x}_1 \vee \bar{x}_2$ is one of the clauses to be satisfied and variable x_1 is set to `true`, then variable x_2 will necessarily have to be assigned a `false` value.

Considering the above observations, we propose a second rounding procedure, called scheme *type B*. This scheme has two possible outcomes. Either it produces a satisfiable truth assignment and hence indicates that the interior point algorithm should halt, or it does not find a satisfiable truth assignment, indicating that the interior point algorithm should proceed. Rounding scheme B can be described by the following five step procedure.

STEP 0: Sort all interior point variables w_j, $j = 1, \ldots, m$, in decreasing order of their absolute values and place the sorted variables in a priority queue

$$Q = \{w_{j_1}, w_{j_2}, \ldots, w_{j_m}\},$$

where

$$|w_{j_1}| \geq |w_{j_2}| \geq \cdots \geq |w_{j_m}|.$$

STEP 1: Select the first variable from Q, say w_k and set

$$\tilde{w}_k = \begin{cases} +1 & \text{if } w_k > 0 \\ -1 & \text{if } w_k \leq 0. \end{cases}$$

Remove variable w_k from Q.

STEP 2: The boolean variable x_k, corresponding to w_k is assigned the appropriate value, i.e.

$$x_k = \begin{cases} \texttt{true} & \text{if } \tilde{w}_k = +1 \\ \texttt{false} & \text{if } \tilde{w}_k = -1. \end{cases}$$

STEP 3: Consider all clauses where x_k occurs, negated or otherwise. As a consequence of the assignment of the truth value to x_k some of these clauses may become satisfied, while others may be reduced. If there is any single literal clause in the reduced set of clauses, that boolean variable is forced to take on a definite value. For example, if $C : \bar{x}_1$ is a single literal clause, x_1 is forced to be `false`. Remove that variable from Q. Set the boolean variable to its required value. As a consequence, some clauses may be made satisfiable, some may be reduced. Furthermore, there may exist two clauses for which a variable must take on contradicting values. For example, if $x_1 =$ `false` in $C_1 : x_1 \vee x_2$ and $C_2 : x_1 \vee \bar{x}_2$, then any value given to x_2 will cause either C_1 or C_2 to become unsatisfiable. If a contradiction is found, terminate the rounding procedure, indicating that a satisfiable truth assignment has not been found and proceed with the interior point algorithm. Note that the current state w^k of the interior point algorithm is not affected if the rounding scheme fails to produce a satisfiable truth assignment. STEP 3 is recursively executed until there are no more single literal clauses to process.

STEP 4: If all clauses are satisfied, terminate the rounding procedure and the integer programming algorithm with a satisfiable truth assignment. Else, go to STEP 1.

3.3. Local Minima

The integer programming algorithm is not guaranteed to converge to a satisfiable solution. When the algorithm converges to a local minimum that is not global one could do as in [14] or [15]. We have not yet implemented any such scheme for this class of problems and therefore simply stop and say the algorithm failed to find a feasible truth assignment.

4. Computational Results

We now describe the computational experiments. We have generated hundreds of random instances of SAT using a model similar to the one described by Hooker [9] [10].

As input to the random instance generator we provide the number of clauses n, the number of variables m and the expected number of literals per clause k. Elements a_{ij} of the constraint matrix A are assigned value $+1$ with probability $p/2$, -1 with probability $p/2$ and 0 with probability $1-p$, where p is such that the expected number nonzero elements per clause is k. All null clauses are rejected. We generate no single literal clauses, differing here from [9], since those clauses can be trivially removed. By removing single literal clauses simple linear programming relaxation can no longer be used to prove unsatisfiability. In [10], Hooker also discards single literal clauses.

All runs were carried out on *mhkorbx*, a KORBX(R) parallel/vector computer. The KORBX(R)

uses a variant of the UNIX(R) Operating System, called the KORBX(R) Operating System. The KORBX(R) machine we used operates in scalar mode at approximately 1 MFlops and at 32 MFlops with full vector concurrent mode. In our experiments all code was written in FORTRAN and C and compiled on the KORBX(R) `fortran` compiler with optimization flags `-O -DAS` and KORBX(R) `cc` compiler with optimization flag `-O`. No special care was taken to vectorize or parallelize the code. All times reported are user times given by the system call `times()`.

We report for each problem class the number of instances proven satisfiable, the number of instances in which the algorithm converged to a local minimum, the minimum, average and maximum number of iterations and the minimum, the minimum, average and maximum solution time of the integer programming algorithm.

We used the following algorithm parameter settings for all problem instances: $\gamma_0 = 10$, $\underline{l}_0 = 0.5$, $\overline{l}_0 = 1.0$, $\epsilon = 10^{-12}$, $\overline{l}_r = 4$, $\alpha = 0.5$, $\gamma_r = \sqrt{2}$ and $l_r = 4$. Our implementation uses a preconditioned conjugate gradient algorithm to solve (19) at each iteration. The conjugate gradient algorithm stops when $|1 - \cos\theta| < 10^{-8}$, where θ is the angle between $M\Delta w$ and h.

Table 4.1 summarizes the computational results for the integer programming algorithm using rounding scheme A. There, results are tabulated for instances varying from 50 variables by 100 clauses, with an average of 5 literals per clause to 1,000 variables by 2,000 clauses and an expected number of 15 literals per clause. Statistics for iterations and solution times include only instances in which the algorithm converged to a global minimum. Times are in seconds.

problem			instances		iterations			solution time		
vars	clauses	$E[n(\mathcal{C})]$	global min	local min	min	mean	max	min	mean	max
50	100	5	89	11	1	46.2	530	0.5	9.4	67.0
100	200	5	82	18	1	136.2	1002	1.0	50.9	297.3
200	400	7	81	19	1	412.6	2070	2.3	313.7	1450.7
400	800	10	86	14	1	44.2	1320	4.8	63.6	1577.9
500	1000	10	80	20	1	119.4	2350	6.0	230.9	4464.5
1000	2000	10	6	4	1	1450.0	3374	18.2	4021.2	8183.2
1000	2000	11	8	2	1	174.6	659	13.6	715.5	2032.1
1000	2000	12	9	1	1	194.9	1535	13.6	571.0	4052.0
1000	2000	13	10	0	1	78.0	7710	14.7	215.4	2008.0
1000	2000	15	9	1	1	1.0	1	15.0	16.5	19.8

Table 4.1 – Computational results (rounding scheme A)

We can make the following observations about the runs in Table 4.1:

- The interior point method produced satisfiable truth assignments for the majority of instances.
- All problem classes (rows in the table) had at least one instance for which the algorithm produced a satisfiable solution in one iteration.

KORBX and UNIX are registered trademarks of AT&T.

- For the problems with $m = 1,000$ variables and $n = 2,000$ clauses, the interior point algorithm produces satisfiable truth assignments in more number of cases as the expected number of literals per clause increases, i.e. as the problem becomes easier.

We reran all instances in which the integer programming algorithm with rounding scheme A took more than one iteration, this time using rounding scheme B. We also ran larger and comparatively more difficult problems with this scheme. These had 1,000 variables and had from 2,000 to 32,000 clauses. Those results are shown in Table 4.2.

problem			instances		iterations			solution time		
vars	clauses	$E[n(C)]$	global min	local min	min	mean	max	min	mean	max
50	100	5	52	0	1	1.7	35	0.5	0.7	7.7
100	200	5	70	0	1	1.0	1	1.0	1.1	1.7
200	400	7	69	0	1	1.0	1	2.4	3.5	6.3
400	800	10	31	0	1	1.0	1	5.1	5.6	7.7
400	800	7	20	0	1	1.0	1	4.7	7.8	16.1
500	1000	10	49	0	1	1.0	1	6.5	7.4	9.8
1000	2000	10	10	0	1	1.0	1	14.9	18.5	20.5
1000	2000	7	50	0	1	1.0	1	18.1	21.5	27.3
1000	2000	3	49	1	1	310.0	7171	8.7	420.7	9595.5
1000	4000	10	10	0	1	1.0	1	24.0	25.1	25.8
1000	4000	4	1	1	3398	3398.0	3398	8314.3	8314.3	8314.3
1000	8000	10	10	0	1	1.0	1	37.5	38.0	38.8
1000	16000	10	10	0	1	1.5	6	46.2	66.4	92.1
1000	32000	10	10	0	1	24.4	235	80.1	232.4	311.3

Table 4.2 – Computational results (rounding scheme B)

We can make the following observations about the runs in Table 4.2:

- Scheme B went to a non-integer solution in very few (only 2) cases .
- The majority of instances were resolved in only one iteration.
- All ten instances of size $m = 1,000$ variables and $n = 32,000$ clauses were resolved in less than 6 CPU minutes.
- All problem classes (rows in the table), with the exception of ($m = 1,000, n = 4,000, E[n(C)] = 4$) had instances that were solved in a single iteration. Only two instances were tested for the class that had no single iteration solution.

In [9], Hooker reports that by solving a linear programming relaxation of SAT one frequently produces an integer solution. We have used MINOS 5.1 [18] to solve the linear programming relaxation

$$\text{minimize } z_0$$

subject to:

$$z_0 + \sum_{j \in J_{c_i}} z_j - \sum_{j \in J_{c_i}} z_j \geq 1 - |J_{c_i}|, \quad i = 1, \ldots, n$$

$$0 \leq (z_0, z_j) \leq 1, \quad j = 1, \ldots, m$$

corresponding to some of the smaller problems tested on the integer programming algorithm. Table 4.3 summarizes the results.

problem			instances		iterations			solution time		
vars	clauses	$E[n(C)]$	int	frac	min	mean	max	min	mean	max
50	100	5	23	77	46	93.4	139	1.62	3.52	5.52
50	100	10	93	7	61	131.5	219	2.6	5.8	10.0
100	200	5	11	89	157	355.1	574	10.7	24.7	41.0
200	400	7	23	77	1011	1523.5	2042	142.9	231.1	316.2
400	800	10	2	3	4799	5610.0	6593	1285.9	1790.0	2147.0
400	800	7	0	15	4982	6143.3	7486	1616.3	2050.4	2802.7

Table 4.3 – Computational results – linear programming relaxation

We can make the following observations about the runs in Table 4.3:

- Contrary to Hooker [9], where single literal clauses were admitted, the simplex method failed to find integral solutions to the linear programming relaxations in the majority of instances tested. In fact, for $n = 400$ variables, $m = 800$ clauses and $E[n(C)] = 7$ literals per clauses, the simplex method failed in all 15 instances. To achieve the performance reported by Hooker for ($n = 50$, $m = 100$, $E[n(C)] = 5$), we had to double $E[n(C)]$ to 10.
- Large instances could not be solved because of excessive solution times.
- The average solution time ratios between MINOS and the interior point method (using rounding scheme B) are shown in Table 4.4.

problem			cpu time
vars	clauses	$E[n(C)]$	ratios
50	100	5	5.0
100	200	5	22.5
200	400	7	66.0
400	800	7	262.9
400	800	10	319.6

Table 4.4 – Computational results – CPU time ratios

Because of these time ratios, we feel that simplex based branch and cut methods will difficultly outperform this interior point approach, since at least one linear programming relaxation must be solved.

5. Concluding Remarks

An interior point algorithm for integer programming [12] [13] has been applied to find truth assignments in instances of SAT. The algorithm finds truth assignments in the majority of large instances (up to 1,000 variables and 32,000 clauses) with the straightforward rounding scheme A and takes very few iterations with the more involved rounding scheme B. We cannot, however, guarantee that the algorithm will find a truth assignment if one exists. Moreover, we cannot make any conclusion when the algorithm converges to a local minimum without finding a satisfiable truth assignment.

Large instances of SAT were solved in little CPU time. In general, MINOS 5.1 required more CPU time to solve only a linear programming relaxation of the problem, in which fractional solutions were frequently found.

In this implementation of the interior point algorithm, no special treatment was given to local minima. We have observed in the computational experiments described in this paper, that when a local minima is encountered, it is usually the case that very few constraints remain unsatisfied. Future work will focus on treating local minima. One possible approach is to add cuts and restart the algorithm, as in [14]. Another is to use another potential function or a branching scheme, as is done in [15]. The question whether one can prove nonsatisfiability with local minima information remains open.

In light of the results presented here, the interior point approach can serve as an efficient tool in solving instances of SAT that before were considered unsolvable simply because SAT is NP-complete.

References

[1] C.E. Blair, R.G. Jeroslow, and J.K. Lowe. Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research*, 5:633–645, 1986.

[2] T.M. Cavalier, P.M. Pardalos, and A.L. Soyster. Modeling and integer programming techniques applied to propositional calculus. *Computers and Operations Research.* To appear.

[3] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[4] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[5] P.L. Hammer and S. Rudeanu. *Boolean methods in operations research and related areas.* Springer-Verlag, 1968.

[6] P. Hansen, B. Jaumard, and M. Minoux. Algorithm for deriving all logical conclusion implied by a set of boolean inegralitier. *Mathematical Programming*, 34:223–231, 1986.

[7] J.N. Hooker. A quantitive approach to logical inference. *Decision Support Systems*, 4:45–69, 1988.

[8] J.N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12:217–239, 1988.

[9] J.N. Hooker. Resolution vs. cutting plane solution of inference problems: Some computational experience. *Operations Research Letters*, 7(1):1–7, 1988.

[10] J.N. Hooker and C. Fedjki. Branch-and-cut solution of inference problems in propositional logic. Technical Report 77-88-89, GSIA, Carnegie Mellon University, Pittsburgh, PA 15213, August 1989.

[11] R.G. Jeroslow. Computation-oriented reductions of predicate to propositional logic. *Decision Support Systems*, 4:183–197, 1988.

[12] N. Karmarkar. An interior-point approach to NP-complete problems – extended abstract. In *Mathematical developments arising from linear programming algorithms.* Summer Research Conference sponsored jointly by AMS, IMS and SIAM. Bowdoin College, Brunswick, Maine, June 1988.

[13] N. Karmarkar, M.G.C. Resende, and K.G. Ramakrishnan. An interior-point algorithm for zero-one integer programming. In *The 13th International Symposium on Mathematical Programming.* Mathematical Programming Society, Tokyo, September 1988.

[14] N. Karmarkar, M.G.C. Resende, and K.G. Ramakrishnan. An interior point algorithm to solve computationally difficult set covering problems. Technical report, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974, 1989.

[15] N. Karmarkar, M.G.C. Resende, and K.G. Ramakrishnan. An interior point approach to the maximum independent set problem in dense random graphs. Technical report, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974, 1989.

[16] K. Levenberg. A method for the solution of certain problems in least squares. *Quart. Appl. Math.*, 2:164–168, 1944.

[17] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11:431–441, 1963.

[18] B.A. Murtagh and M.A. Saunders. MINOS 5.0 user's guide. Technical Report SOL 83-20, Dept. of Operations Research, Stanford University, Stanford, CA, 1983.

[19] N.J. Nilson. *Principles of artificial intelligence.* Tioga, 1980.

[20] S. Sahni. Computationally related problems. *SIAM Journal of Computing*, 3:262–279, 1974.

[21] H.P. Williams. Logical problems and integer programming. *Bulletin of the Institute of Mathematics and its Applications*, 13:1820, 1977.

[22] H.P. Williams. Linear and integer programming applied to the propositional calculus. *Systems Research and Information Sciences*, 2:81–100, 1987.

[23] R.R. Yager. A mathematical programming approach to inference with the capability of implementing default rules. *International Journal of Man-Machine Studies*, 29:685–714, 1988.

An Interior-Point Approach to NP-complete Problems

Narendra Karmarkar

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

In this paper, we extend the interior-point approach used in Karmarkar's linear programming algorithm to solve NP-complete problems formulated as quadratic optimization problems over polytopes. We associate a potential function with the quadratic objective function and create a sequence of interior points to minimize the potential function. At each point in the sequence we optimize a local quadratic approximation to the potential function, over a search region which is similar to the search region used in linear programming.

We then define a class of functions more general than convex functions having path-wise connected level sets and prove that functions in this class do not have any spurious local minima (i.e. those local minima which are not global minima). We prove that a particular family of potential functions used in our approach belongs to this class of functions.

1. Introduction

As a representative of NP-complete problems, we study the following integer programming problem:

IP Problem: Find a feasible solution $\underline{x} \in \mathbf{R}^n$ such that

$$A\underline{x} \leq \underline{b}$$

$$\text{and } x_i \in \{-1, 1\} \ \forall i = 1, \ldots, n$$

$$\text{where } A\text{: is an } m \times n \text{ real matrix and } b \in \mathbf{R}^m .$$

The more common form of integer programming which requires each variable x_i to take 0-1 values can be easily converted to the above form by the substitution $x_i = \frac{1 + x_i'}{2}$, so that x_i' take ± 1 values.

Consider the following linear programming relaxation of the integer programming problem by allowing each variable x_i to take values in the closed interval [-1,1].

LP Problem: Find $\underline{x} \in \mathbf{R}^n$ such that

$$A\underline{x} \leq \underline{b}$$

$$-1 \leq x_i \leq 1$$

Solution of the LP relaxation can be useful in solving the IP problem in several ways:

1. If the LP problem is infeasible, then one can infer that the IP problem is also infeasible.
2. In a branch-and-bound type of method for solving IP [4], if the LP relaxation of the reduced problem obtained after partial assignment of ± 1 values to a subset of the variables is infeasible, then that partial solution can not be extended to a feasible solution of the integer programming problem.
3. If a vertex of the polytope associated with the LP problem happens to have ± 1 coordinates, then it is also a solution to the integer programming problem. Many unwanted fractional solutions (including zero-valued solutions in case of ± 1 formulation) can be ruled out by augmenting the linear program with additional inequalities. An approach based on polyhedral combinatorics seeks to generate several such families of inequalities [9].

Further selectivity in the type of solution to the linear programming problem can be gained by using a quadratic objective function, $\sum_{i=1}^{n} x_i^2$, which prefers integral solutions over fractional ones, thus making it unnecessary to explicitly rule out fractional solution with small values of the quadratic objective and achieving economy in the number of inequalities needed to obtain integral solution. In this paper, we elaborate on an interior-point approach based on quadratic objective function. In order to test our approach, we choose problem classes such as finding maximum independent set in dense random graphs and finding minimum width of Steiner triple systems, in which fractional solutions to the linear programming relaxation are abundant and it is very hard to find integral solutions. In 0-1 formulations of these problems, most coordinates have values 1/2 and 1/3 respectively. For both problems, lower bounds on combinatorial algorithms have been proved in restricted models of computation which grow like e^{kn} and $e^{k'\sqrt{n}}$ respectively [2], [1], [3]. For the latter problem, the model used is more general because it allows linear programming relaxation to be solved at each node of the branch-and-bound method. Hence these problem classes are good test cases to see additional selectivity gained by quadratic objective function using interior-point approach. We have also tested our approach on the satisfiability problem in mathematical logic. Reports on experimental results on these problem classes are the subject of a forthcoming papers [5], [11].

However, many real-life applications of integer programming are much simpler. Our method has already been applied to developing global router for VLSI design [10]. The method described in this paper is part of an ongoing research program in interior-point approach. Rather than describing a single algorithm, we have described a general approach for constructing interior-point algorithms for difficult combinatorial problems. We believe that efficient algorithms for many difficult combinatorial problems can be constructed on the basis of a common principle similar to what we have outlined in this paper and these algorithms will differ mainly in the way additional structure of the particular problem class is exploited to gain further computational efficiency. Detailed description of the algorithm at the level of pseudo-code as applied to any specific problem class will be included in the paper describing experimental results on that particular problem class.

Now we relate the following quadratic programming problem to the integer programming problem.

QP Problem: Maximize

$$\sum_{i=1}^{n} x_i^2 , \qquad x \in \mathbf{R}^n$$

subject to:

$$Ax \leq b$$

$$-1 \leq x_i \leq 1$$

Note that for any feasible solution to the quadratic programming problem, $|x_i| \leq 1 \Rightarrow \sum_{i=1}^{n} x_i^2 \leq n$ Hence the maximum value of objective function is at most n. If x is a feasible solution to IP, then it is a feasible solution to QP and $x_i \in \{-1, 1\} \Rightarrow \sum_{i=1}^{n} x_i^2 = n$, hence it is an optimal solution to QP with optimal value n.

Conversely any optimal solution to QP with objective value n is a feasible solution to the IP problem.

The problem transformation given above is by itself of little value since the new problem is also NP-complete and hopelessly difficult computationally, if one were to employ an extreme-point approach as in the simplex method, to solve it. What makes the problem transformation computationally useful is the recent insight into the geometry of linear programming polytopes gained by interior point-approach such as Karmarkar's algorithm [5]. As a by-product of this approach we discovered, for the first time, a method of making good spherical approximation to a polytope, around any strictly interior-point in the following sense.

Given any point in the strict interior of the polytope, there is a projective transformation of the space, so that in the transformed space it is possible to draw two spheres, circumscribing and inscribing the polytope, both centered at the image of the given point and having a "small" ratio of radii. In the worst-case the ratio can be guaranteed to be no more than n, but in the average-case it is even smaller.

Secondly, optimizing a quadratic function over a ball or an ellipsoid is a computationally easy problem, even if the function is not convex [7]. We have included a discussion of this problem in the Appendix, for reader's convenience.

2. Outline of the Continuous Minimization Approach

For a feasible problem, the optimal value of $\sum_i x_i^2$ over the feasible region is n. Using this information, we can obtain the two forms of objective function we have used most often, called "spherical form" and "conical form", both having optimal value zero.

$$f(x) = \sqrt{n - \sum_i x_i^2} \quad \text{(spherical form)}$$

$$f(x) = \sqrt{n + \delta} - \sqrt{\sum_i x_i^2 + \delta} \quad \text{(conical form)}$$

Here δ is some positive constant, used to avoid nondifferentiability of the square-root function when $\sum_i x_i^2 = 0$.

In case of the projective form of the problem given later, it is easy to make $f(x)$ homogeneous by using the relation $\sum_i x_i = 1$. For example, the spherical form can be written as

$$f(x) = \sqrt{n \cdot (\sum_i x_i)^2 - \sum_i x_i^2}$$

If we want to impose equality constraints of the type $A\underline{x} = \underline{b}$ on the solution, their effect can be incorporated into the quadratic objective function $f(x)$ by changing it to $f(x) + \|Ax - b\|^2$. In case of feasible problem, optimal value continues to be zero after this modification. We assume that such a modification has been done in stating the standard problem forms below, although equivalent problem formulations with explicitly imposed equality constraints are also possible.

From the objective function, we construct the associated potential function, just as in linear programming.

Let $s_i(\underline{x})$, $i = 1, 2, \ldots, N$ denote the slack variables which are linear functions of $\underline{x}$. These include slack variables from upper and lower bound constraints $-1 \leq x_i \leq 1$ as well as slack variables from other inequality constraints $A\underline{x} \leq \underline{b}$.

Then the potential function $g(x)$ associated with an objective function $f(x)$ is given by

$$g(\underline{x}) = \frac{(f(x))^N}{\prod_{i=1} s_i(\underline{x})}$$

A more general class of potential functions is based on non-negative weights $\{w_i\}$:

$$g_w(x) = \frac{(f(x))^{(\sum_i w_i)}}{\prod_{i=1}^{N} s_i^{w_i}(x)}$$

Suppose the potential function is re-expressed as a function of nonnegatively constrained variables $s_1, s_2, \ldots, s_N$.

Then the simplest projective form of the problem is:

$$\text{minimize} \quad g(s_1, s_2, \ldots, s_N)$$

$$\text{subject to}, \quad s_i \geq 0, \quad i = 1, \ldots, N$$

$$\text{and} \sum_{i=1}^{N} s_i = 1$$

and the affine form is given by

$$\text{minimize } g(s_1, s_2, \ldots, s_N)$$

$$\text{subject to } s_i \geq 0, \quad i = 1, \ldots, N.$$

Now we outline the general approach used, which is quite similar to the interior-point algorithms for linear programming:

1. Obtain an initial strictly interior solution to be used as the starting point. For many combinatorial problems, there is usually a trivial way of getting such a starting point. A more general method is to use an interior-point linear programming algorithm to obtain the starting point. In this case the starting point is also the center of the polytope, in the

sense of maximizing product of the distances from hyperplanes defining the polytope.

2. At each major iteration of the algorithm, transform the problem by projective or affine transformation to put the current point at the center, and construct the inscribed sphere. Apply the corresponding transformation to the potential function.

3. Construct a Taylor series expansion of the transformed potential function and retain terms up to the quadratic term. (If only expansion up to the linear term is used, the resulting method can be rather easily incorporated into a linear programming system, by augmenting it with a subroutine which supplies the gradient of the potential function to be used at each iteration in place of a fixed linear objective function. The resulting method can still be effective on simple real-world problems, although, in general it will not perform as well as the quadratic method on harder problems.)

4. Optimize the quadratic approximation over the inscribed sphere. In general, it is possible that the new point computed this way is worse (in terms of potential function value) than the current point due to the third or higher order terms which we ignored. In such case radius of the search region must be reduced repeatedly until the optimization based on quadratic approximation improves the current solution or the radius of the search region is smaller than certain tolerance ε. In the latter situation, we regard the current point as a local minimum. From a theoretic point of view, we consider a point to be local minimum if the gradient of the potential function is zero and all eigenvalues of the hessian are nonnegative. Discussion of the treatment of local minimum is given in a latter section.

   Minimization of a local approximation to the function over a search region is analogous to a "trust region" approach. However, there is a major difference. While in a trust region approach, the choice of trust region is based on *local* considerations, in our approach the choice of inscribed ellipsoid as a search region is based on the fact that such a region provides a provably good *global* approximation to the polytope as shown in the proof of polynomial time behavior of interior-point approach to linear programming [6].

5. Assuming that local minima is not reached, apply the inverse of the centering transformation to obtain the next point in the sequence. Then we round the solution to the nearest ± 1 solution, or apply other rounding techniques described in a later section. If the rounded point satisfies all the constraints, the procedure terminates, otherwise we repeat steps 2 to 5, from the new (unrounded) interior point.

3. Analysis of Connectivity Properties

3.1 Local Minima in Continuous Minimization Procedures:

Throughout this section, f is a $\mathbf{C}^3$ function from an open subset $U \subseteq \mathbf{R}^n$ to $\mathbf{R}$.

Let $\underline{g}(\underline{x})$ and $H(\underline{x})$ denote the gradient vector and the Hessian matrix of the function f at a point $x \in U$. i.e.

$$g_i(\underline{x}) = \frac{\partial f}{\partial x_i}(\underline{x}), \quad i = 1, \ldots, n$$

and

$$H_{ij}(\underline{x}) = \frac{\partial^2 f}{\partial x_i \partial x_j}(\underline{x}), \quad i, j = 1, , \ldots, n$$

Definition:

A point $\underline{x} \in U$ is called a *local minimum* of the function f if

$$g_i(\underline{x}) = 0 \quad i = 1, \ldots, n$$

and

$$H(\underline{x}) \quad \text{is positive definite}$$

It is well known that a first-order method such as steepest decent method and a second-order method such as Newton's method for function minimization converge to a local minimum when started sufficiently close to it.

Definition:

A point $\underline{x} \in U$ is called a *global minimum* of the function f over U if for all $y \in U$,

$$f(\underline{y}) \geq f(\underline{x})$$

Definition:

A point $\underline{x} \in U$ is called a *spurious local minimum* of the function f over V if it is local minimum of f but not a global minimum of f over U.

The steepest decent method and Newton's method can converge to a spurious local minimum and thus fail to find a global minimum.

3.2 Importance and Limitations of Convexity

If the function $f(\underline{x})$ being minimized is convex, then we can rule out the possibility of having spurious local minima a priori. This property makes the class of convex functions rather important in the study of function minimization methods and has attracted the due attention from many researchers. Convex functions (with further suitable restrictions) can be minimized in polynomial-time [ ], [ ].

This leads to a natural question: Is convexity the characteristic property that separates the class of efficiently solvable minimization problems from the rest? Is there a broader class of functions for which existence of spurious local minima can be ruled out a priori?

Here we give at least two reasons why a broader class is needed:

Even if the given function $f(x)$ is not convex, it may be possible to define a Riemannian metric on the domain U such that for every pair points in U, there is a unique geodesic joining them and levels sets of f are geodesically convex in the sense that for every pair of points in a level set, the geodesic joining them lies entirely in the same level set.

Secondly, if the given function is convex and we apply a non-linear transformation to the space which is a diffeomorphism, the transformed function need not be convex, but the transformed problem may still be easy in spite of non-convexity. In fact if there is a system of continuous trajectories defined in the original space which allow us to construct an algorithm for the global minimization problem, the image of these trajectories under the transformation gives us a way of solving the transformed problem. This argument suggests that the class of efficiently solvable continuous optimization problems should be larger than the class of convex problems, and the property characterizing the broader class should be topologically invariant.

In the next subsection we identify a class of functions, more general than convex functions, in terms of the connectivity of the levels sets of the function.

In the following subsection, we give a formulation of NP-complete problems as a continuous optimization problems belonging to this class.

3.3 Importance of Connectivity:

Definition:

A subset $X \subseteq R^n$ is said to be pathwise connected if for every pair of points $\underline{a}, \underline{b} \in X$, there exists a continuous path in X joining the two points. i.e. there exists a continuous function $\underline{g}(t) : [0, 1] \rightarrow X$ such that $\underline{g}(0) = \underline{a}$ and $\underline{g}(1) = \underline{b}$.

Definition:

Let $f(\underline{x})$ be a function from an open set $U \subseteq R^n$ to R. Level set of f corresponding to a level $\alpha \in R$ is denoted by $L_f(\alpha)$ and is defined as

$$L_f(\alpha) = \{\underline{x} \in U | f(\underline{x}) \leq \alpha\} .$$

In the following theorem we explain the importance of the class of functions having connected level sets.

Theorem:

Let $f(\underline{x})$ be a $\mathbf{C}^3$ function from an open set $U \subseteq R^n$ to R. Suppose the level sets $L_f(\alpha)$ of f are pathwise connected for all values of α. Then the function f can not have a spurious local minimum.

Proof:

If possible let $\underline{x}_0 \in U$ be a spurious local minimum of the function f. Since $\underline{x}_0$ is a local minimum, it is possible to choose a non-linear co-ordinate system so that f can be expressed as sum of squares near $\underline{x}_0$, by applying the reduction theorem from Morse theory [8] stated below:

If $f(\underline{x})$ is a $\mathbf{C}^3$ function from an open set $U \subseteq R^n$ to R and $\underline{x}_0 \in U$ is a local minimum of f, then there exists an open ball $B(\underline{x}_0, \varepsilon) \subseteq U$ of radius $\varepsilon > 0$ centered around $\underline{x}_0$ and a one-to-one and onto $\mathbf{C}^1$ map

$$\phi(x) : B(\underline{x}_0, \varepsilon) \rightarrow S \subseteq R^n$$

having a $\mathbf{C}^1$ inverse such that

$$\phi(\underline{x}_0) = \underline{0}$$

$$\text{and} \quad f(\phi^{-1}(\underline{y})) = f(\underline{x}_0) + \sum_{i=1}^{n} y_i^2 \quad \text{for all} \quad \underline{y} \in S .$$

If $\underline{x}' \in B(\underline{x}_0, \varepsilon)$ and $\underline{x}' \neq \underline{x}$, then $\underline{y} = \phi(\underline{x}') \in S$ and $\underline{y} \neq 0$. Hence $f(\underline{x}') = f(\phi^{-1}(\underline{y})) > f(\underline{x}_0)$.

Since $\underline{x}_0$ is not a global minimum, there exists a point $\underline{z} \in U$ such that $f(\underline{z}) < f(\underline{x}_0)$. Let L denote the level set $L_f(f(\underline{x}_0))$. Clearly $\underline{x}_0, \underline{z} \in L$. Since L is pathwise connected, there exist a function

$$\underline{g}(t) : [0, 1] \rightarrow L$$

such that $\underline{g}(0) = \underline{x}_0$ and $\underline{g}(1) = \underline{z}$. Let $\underline{r}(t) : [0, 1] \rightarrow R$ be defined as

$$r(t) = \| \underline{g}(t) - \underline{x}_0 \| .$$

Clearly $r(t)$ is a continuous function. Let $r_{\max} = \max_{t \in [0,1]} r(t)$. Therefore $r(t) : [0, 1] \rightarrow [0, r_{\max}]$. Let $r' = \frac{1}{2} \min \{\varepsilon, r_{\max}\}$ and $r' \neq 0$. Hence there exists a $t' \in (0, 1]$ such that $r(t') = r'$. Let $\underline{x}' = \underline{g}(t')$ since $g(t) \in L$ for all $t \in [0, 1]$, $\underline{x}' \in L$. Hence $f(\underline{x}') \leq f(\underline{x}_0)$. But $\underline{x}' \in B(\underline{x}_0, \varepsilon)$, and $\underline{x}' \neq \underline{x}_0$ hence $f(\underline{x}') > f(\underline{x}_0)$, thus obtaining a contradiction.

3.4 Connectivity Properties for NP-complete Problems

In case of minimization of convex quadratic function over a polytope, the level sets are connected and there is a polynomial-time algorithm for solving the problem. In contrast, minimization of a concave quadratic function over a polytope is NP-complete, and the level sets of such a function can have an exponential number of connected components e.g. let the polytope be defined by

$$-1 \leq x_i \leq 1 \qquad i = 1, \ldots, n .$$

Let the concave function be

$$f(\underline{x}) = n - \Sigma x_i^2$$

and take the value of level α such that $0 < \alpha < 1$. Since each vertex of the polytope is in this level set and is disconnected from all other vertices, the number of connected components is exponential.

However, in our approach, we associate a "potential function" with the objective function. The level sets of potential function appears to have far fewer connected components than those of the objective function. The following theorem offers a partial explanation of this phenomenon by showing that for a particular class of potential functions and homogeneous form of the problem, there is only one connected component. The problem is defined as follows

Problem: Given a system of homogeneous linear equations

$$A\,\underline{x} = 0, \quad \text{where} \quad A: \; m \times n \quad \text{matrix of real numbers}$$

$$\text{and} \quad \underline{x} \in R^n .$$

Find a solution $\underline{x}$ with each component

$$x_i = +1 \quad \text{or} \quad -1 \qquad i = 1, \ldots, n .$$

Note that there is no loss of generality in assuming the homogeneity. This problem too is NP-complete.

The simplest form of the potential function to which the theorem applies is

$$f(x) = \frac{\left[n - \sum_{i=1}^{n} x_i^2\right]^n}{\prod_{i=1}^{n} (1 - x_i^2)} .$$

A more general form of the potential function to which the theorem applies is defined in terms of a set of weights $\{w_i \mid i = 1, \ldots, n\}$ such that $w_i \geq 0$ and at least one $w_i > 0$.

$$f(x) = \frac{\left[\sum_{i=1}^{n} w_i (1 - x_i^2)\right]^{\left[\sum_{i=1}^{n} w_i\right]}}{\prod_{i=1}^{n} (1 - x_i^2)^{w_i}}$$

The level set corresponding to level α is

$$L(\alpha) = \{\underline{x} \mid \quad A\,x = 0, \quad -1 \le x_i \le 1, \quad f(x) \le \alpha\} .$$

Theorem: The level set $L(\alpha)$ is pathwise connected for each α.

Proof: Let $\underline{x}$ be any feasible point i.e.

$$A\,\underline{x} = 0, \quad -1 \le x_i \le 1$$

By definition of level sets, $\underline{x} \in L(f(\underline{x}))$. Note that $-\underline{x}$ is also feasible and $f(-\underline{x}) = f(\underline{x})$ hence $-\underline{x} \in L(f(\underline{x}))$. We first show that the entire straight-line segment connecting $\underline{x}$ to $-\underline{x}$ is in $L(f(\underline{x}))$. Let $\underline{u} = \alpha\,\underline{x}$ be any point on this segment, so that $|\alpha| \le 1$. Then $A\underline{u} = 0$ and $-1 \le u_i \le 1$, hence $\underline{u}$ is feasible. To show that $f(\underline{u}) \le f(\underline{x})$ we define

$$g(\alpha) = \ln (f(\alpha\,\underline{x}))$$

$$g(\alpha) = \left[\sum_{i=1}^{n} w_i\right] \ln \left[\sum_{i=1}^{n} w_i (1 - \alpha^2 x_i^2)\right] - \sum_{i=1}^{n} w_i \ln (1 - \alpha^2 x_i^2) .$$

Hence

$$g'(\alpha) = -\frac{2\alpha \cdot \left[\sum_{i=1}^{n} w_i\right]\left[\sum_{i=1}^{n} w_i x_i^2\right]}{\sum_{i=1}^{n} w_i (1 - \alpha^2 x_i^2)} + 2\alpha \cdot \sum_{i=1}^{n} \frac{w_i x_i^2}{1 - \alpha^2 x_i^2} .$$

It is easy to rewrite $g'(\alpha)$ in the following form

$$g'(\alpha) = \frac{1}{\alpha} \cdot \frac{1}{\sum_{i=1}^{n} w_i (1 - \alpha^2 x_i^2)} \sum_{i,j=1}^{n} w_i w_j \left[\sqrt{\frac{1 - \alpha^2 x_i^2}{1 - \alpha^2 x_j^2}} - \sqrt{\frac{1 - \alpha^2 x_j^2}{1 - \alpha^2 x_i^2}}\right]^2 .$$

From the expression on the right hand side it is clear that $g'(\alpha) \ge 0$ when $\alpha > 0$ and $g'(\alpha) \le 0$ when $\alpha < 0$.

Therefore

$$g(\alpha) \le g(1) \quad \text{for all} \quad |\alpha| \le 1$$

hence

$$f(\alpha\underline{x}) \le f(\underline{x}) \quad \text{for} \quad |\alpha| \le 1 .$$

Connectivity of $L(f(\underline{x}))$ follows, since we have shown that each point in the level set can be connected to the origin (which belongs to the level set) by a continuous path, in fact by a

straight line.

The above theorem by itself is not enough to allow construction of an algorithm with good theoretical properties, for reasons having to do with the nature of the minimum level set. If $\underline{x}$ is a feasible solution with $x_i = \pm 1$, so is $-\underline{x}$. If there are k such distinct pairs of solutions, denoted by $(\underline{x}^{(1)}, -\underline{x}^{(1)})$, $(\underline{x}^{(2)}, -\underline{x}^{(2)}) \cdots$, $(\underline{x}^{k}, -\underline{x}^{k})$, the minimum level set consists of a union of k straight-line segments, where the i^{th} segment is obtained by joining $\underline{x}^{(i)}$ to $-\underline{x}^{(i)}$. If we find, by means of a minimization procedure, any point in the minimum level set different from the origin, it is easy to obtain the corresponding pair of feasible solutions with ± 1 coordinates. But it is difficult to construct a method that is theoretically guaranteed to avoid the origin. On the other hand practical algorithms based on the concepts described in this paper appear to be succeeding, for reasons which are more complex than what we can explain with the current theoretical understanding of the method. In our empirical study, we have worked with problems having multiple connected components. It is not enough to analyze just the number of connected components, but the ratios of their volumes is also important in the practical success of the approach. The above theorem merely provides an example of a situation in which the level sets of the potential function have drastically fewer connected components than the level sets of the original objective function.

4. Local Minima.

In this section, we describe some simple techniques for treating local minima, while many further improvements are possible in these techniques, they are not so critical in case of problem classes in which the number of local minima encountered is small or zero. Thus many real-world problems falling in this category can already be solved without the need for any sophisticated methods for resolving local minima. As an example, we mention the "global" wire-routing problem in VLSI design [10].

When a local minimum is encountered, we first round the solution to a ± 1 solution by one of these techniques:

1. Rounding to the nearest ± 1 point

$$\text{i.e. if } x_i \geq 0 \text{ set } x_i = 1$$

$$\text{if } x_i < 0 \text{ set } x_i = -1$$

2. When the starting point maximizes the product of slack variables (which is the case when linear programming is used to obtain the starting point), rounding can be based on coordinate-wise comparison of the current solution point within the starting point.

   Let $x^{(0)}$: denote the starting point x : current solution.

   Then

$$\text{set } x_i = 1 \quad \text{if } x_i \geq x^{(0)}$$

$$\text{set } x_i = -1 \quad \text{if } x_i < x_i^{(0)}$$

3. By a technique specific to the combinatorial problem solved, e.g. in the maximum independent set problem, round the largest x_i to $+1$, i.e. include the i^{th} node in the independent set, exclude its neighbors from the graph and repeat.

Let $\underline{w}$ be the rounded solution and suppose it does not satisfy all the constraints. Note that

$$\underline{w}^T x = n \quad \text{for} \quad \underline{x} = \underline{w}$$

and

$$\underline{w}^T \underline{x} \leq n - 2 \ ,$$

for all $\underline{x}$ such that $x_i \in \{-1, 1\}$ and $\underline{w} \neq \underline{x}$.

Thus we can exclude $\underline{w}$ by means of the inequality $\underline{w}^T \underline{x} \leq n - 2$, without converting any other feasible solution to the integer programming problem into an infeasible solution. We augment the problem with this inequality and restart the process from the center of the new polytope. Alternatively, it is also possible to make the current solution feasible w.r.t. the new inequality by means of a few steps of interior-point linear programming algorithm, taking the normal vector to the newly added hyperplane as an objective function. Note that the new inequality does not necessarily eliminate the local minimum in the interior of the polytope, but it removes the point obtained by rounding the local minimum from the feasible region and also modifies the potential function.

APPENDIX

Non-convex Quadratic Optimization Over an Ellipsoid

Consider the following problem:

$$\text{minimize} \quad f(x) = \frac{1}{2} x^t Q x + \underline{g}^T \underline{x} + c$$

$$\text{subject to} \quad \underline{x}^T \underline{x} \leq r^2$$

where Q is an $n \times n$ real symmetric matrix

$$\underline{g}, \underline{x} \in \mathbf{R}^n$$

c, r are real numbers, $r > 0$.

Here, we have taken the search region, $x^T x \leq r^2$ to be a ball of radius r. Note that a quadratic optimization problem defined over an ellipsoid given by $x^T H x \leq r^2$, where H is a positive definite matrix can be converted to the above form by means of a non-singular linear transformation of the space, hence there is no loss of generality in assuming the search region to be a ball.

Since Q is real and symmetric, it has a full set of orthonormal eigenvectors spanning $\mathbf{R}^n$. Let $u_1, u_2, \ldots, u_n$ denote these eigenvectors and let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the corresponding eigenvalues ordered so that $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1} \leq \lambda_n$. Let $\lambda_{\min} = \min_i \{\lambda_i\}$.

Let k = dimension of the eigenspace corresponding to the smallest eigenvalue. Thus $\lambda_1 = \lambda_2 = \lambda_3 \cdots = \lambda_k = \lambda_{\min}$. Then

$$Q = \sum_{i=1}^{n} \lambda_i \underline{u}_i \underline{u}_i^T$$

$$\text{and} \quad I_n = \sum_{i=1}^{n} \underline{u}_i \underline{u}_i^T \text{, where } I_n \text{ is the } n \times n \text{ identity matrix .}$$

If $f(x)$ is a convex function then $\lambda_{\min} \geq 0$, which corresponds to the easy case. Here we assume that $f(x)$ is non-convex i.e. $\lambda_{\min} < 0$.

In order to describe the solution to the above problem, we consider two cases:

Case 1. $\sum_{i=1}^{k} (g^T u_i)^2 > 0$.

Let $\mu \in (-\infty, \lambda_{\min})$ and consider the parametric family of vectors

$$x(\mu) = -\sum_i \frac{(g^T u_i)\, u_i}{\lambda_i - \mu}$$

Then

$$\|x(\mu)\|^2 = \sum_i \frac{(g^T u_i)^2}{(\lambda_i - \mu)^2} .$$

In the range $\mu \in (-\infty, \lambda_{\min})$ each of the terms $\frac{1}{(\lambda_i - \mu)^2}$ is a strictly monotonically

increasing function of μ, each of the numerators $(g^T u_i)^2$ is nonnegative, and at least one of the numerators is strictly positive. Hence $\|x(\mu)\|^2$ is a strictly monotonically increasing function of μ.

Furthermore

$$\lim_{\mu \to -\infty} x(\mu)^T x(\mu) = 0$$

and

$$\lim_{\mu \to \lambda_{\min}} x(\mu)^T x(\mu) = \infty ,$$

since $\exists\ i,\ 1 \le i \le k$, such that $g^T u_i \neq 0$. Hence for any $r > 0$, the equation $x^T(\mu)x(\mu) = r^2$ has a unique solution in μ. We denote this solution by $\mu(r)$.

Then, $x(\mu(r))$ is the unique optimal solution to the above optimization problem.

Proof: Let $\underline{x}'$ be any vector such that $\|\underline{x}'\|^2 \le r^2$

$$f(x) = \frac{1}{2} x^T Q x + g^T x + c$$

$$= \sum_i \frac{1}{2} \lambda_i (x^T u_i)^2 + (g^T u_i)(x^T u_i) + c$$

$$f(\underline{x}') - f(x)$$

$$= \sum \frac{1}{2} \lambda_i [(\underline{x}'^T u_i)^2 - (x^T u_i)^2] + (g^T u_i)[\underline{x}'^T u_i - x^T u_i]$$

$$= \sum \frac{1}{2} (\lambda_i - \mu)[\underline{x}'^T u_i - x^T u_i][\underline{x}'^T u_i + x^T u_i] + (g^T u_i)(\underline{x}'^T u_i - x^T u_i)$$

$$+ \frac{1}{2} \mu [\sum_i (\underline{x}'^T u_i)^2 - (x^T u_i)^2]$$

$$= \sum \frac{1}{2} (\lambda_i - \mu)[\underline{x}'^T u_i - x^T u_i]\{\underline{x}'^T u_i + x^T u_i + 2\frac{g^T u_i}{\lambda_i - \mu}\}$$

$$+ \frac{1}{2} \mu [\|\underline{x}'\|^2 - r^2]$$

but

$$\frac{g'^T u_i}{\lambda_i - \mu} = -x^T u_i$$

$$f(\underline{x}') - f(x) = \sum_i \frac{1}{2} (\lambda_i - \mu)[\underline{x}'^T u_i - x^T u_i]^2 + \frac{1}{2} \mu [\|\underline{x}'\|^2 - r^2]$$

Note that

$$\lambda_i - \mu > 0$$

$$(\underline{x}'^T u_i - x^T u_i)^2 \geq 0$$

$$\|\underline{x}'\|^2 \leq r^2$$

and $\mu < \lambda_{\min} < 0$.

Therefore $f(\underline{x}') \geq f(x)$ proving that x is an optimal solution.

Furthermore if $\underline{x}' \neq x$, then $\exists i : (\underline{x}'^T u_i - x^T u_i)^2 > 0$. Therefore $f(\underline{x}') > f(x)$, proving that x is the unique optimal solution.

Case 2. $g^T u_i = 0 \quad \forall \quad i = 1, \ldots, k.$

Let $\mu \in (-\infty, \lambda_{\min}]$, and consider the parametric family of vectors

$$x(\mu) = -\sum_{i=k+1}^{n} \frac{(g^T u_i) u_i}{\lambda_i - \mu} .$$

Then

$$\|x(\mu)\|^2 = \sum_{i=k+1}^{n} \frac{(g^T u_i)^2}{(\lambda_i - \mu)^2} .$$

Let

$$r_{\max} = \|x(\lambda_{\min})\| .$$

Case 2a. Suppose that r given in the optimization problem is such that

$$r < r_{\max}$$

since $r > 0$, $r_{\max} > 0$. Therefore $\exists\, i > k$ such that $g^T u_i \neq 0$. Hence $\|x(\mu)\|^2$ is a strictly monotonically increasing function of μ in the range $\mu \in (-\infty, \lambda_{\min})$ and the equation

$$\|x(\mu)\| = r \quad \text{has a unique solution in} \quad \mu .$$

We denote this solution by $\mu(r)$.

Then $x(\mu(r))$ is the unique optimal solution to the above optimization problem.

Proof: Similar to Case 1.

Case 2b. Suppose the given $r \geq r_{\max}$.

Let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be any real numbers such that

$$\sum_{i=1}^{k} \alpha_i^2 = r^2 - r_{\max}^2 .$$

Then

$$x = \sum_{i=1}^{k} \alpha_i u_i - \sum_{i=k+1}^{n} \frac{(g^T u_i) u_i}{(\lambda_i - \lambda_{\min})}$$

is an optimal solution.

Proof: Let x' be any vector such that $\|x'\|^2 \le r^2$

$$f(x') = \frac{1}{2} x'^T Q x' + g^T x' + c$$

$$= \frac{1}{2} \sum_{i=1}^{n} \lambda_i (x'^T u_i)^2 + \sum_{i=1}^{n} (g^T u_i)(x'^T u_i) + c$$

$$= \frac{1}{2} \sum_{i=1}^{n} \lambda_{\min} (x'^T u_i)^2 + \frac{1}{2} \sum_{i=1}^{n} (\lambda_i - \lambda_{\min})(x'^T u_i)^2$$

$$+ \sum_{i=k+1}^{n} (g^T u_i)(x'^T u_i) + c$$

$$= \frac{1}{2} \lambda_{\min} \|x'\|^2 + \frac{1}{2} \sum_{i=k+1}^{n} (\lambda_i - \lambda_{\min}) \left[(x'^T u_i)^2 + 2 \left(\frac{g^T u_i}{\lambda_i - \lambda_{\min}} \right) (x'^T u_i) \right] + c .$$

Substituting

$$\frac{g^T u_i}{\lambda_i - \lambda_{\min}} = - x^T u_i , \quad \text{for} \quad i > k ,$$

we get

$$f(x') = \frac{1}{2} \lambda_{\min} \|x'\|^2 + \frac{1}{2} \sum_{i=k+1}^{n} (\lambda_i - \lambda_{\min})[(x'^T u_i)^2 - 2(x'^T u_i)(x^T u_i)] + c.$$

Therefore

$$f(x') - f(x) = \frac{1}{2} \lambda_{\min} [\|x'\|^2 - r^2]$$

$$+ \frac{1}{2} \sum_{i=k+1}^{n} (\lambda_i - \lambda_{\min})(x'^T u_i - x^T u_i)^2$$

since $\lambda_{\min} < 0$, $\|x'\|^2 \le r^2$, $\lambda_i - \lambda_{\min} > 0$ and $(x'^T u_i - x^T u_i)^2 \ge 0$ we have $f(x') \ge f(x)$.

The solution is not unique since the α_i's were chosen arbitrarily.

REFERENCES

[1] Avis, D., "A note on some computationally difficult set covering problems," Mathematical Programming, vol. 18, pp. 138-145, 1980.

[2] V. Chvátal, Determining the Stability Number of a Graph, SIAM Journal of Computing, 6 (1977), pp. 643-662.

[3] Fulkerson, D., Nemhauser, G., Trotter, L. Jr., "Two computational difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems", Mathematical Programming study, vol. 2, pp. 72-81, 1974.

[4] Garfinkel, R., Nemhauser, G., Integer Programming, John Wiley & Sons, 1972.

[5] Karmarkar, N., Ramakrishnan, K. G., Resende, M.G.C., "An Interior-point approach to the maximum independent set problem in dense random graphs" (in preparation).

[6] Karmarkar, N., "A New Polynomial-time Algorithm for Linear Programming, Combinatorica 4, 373-395, 1984.

[7] Moŕe and Sorrensen, SIAM J. Sci. Stat. Comp. 4 (1983), pp. 553-572.

[8] Morse, M., S. Cairns, Critical Point Theory in Global Analysis and Differential Topology, Academic Press, 1969.

[9] Padberg M., Grötschel M., "Polyhedral Computations" in Lawler, Lenstra, Rinnooy Kan, Shmoys (ed.), The Traveling Salesman Problem, John Wiley & Sons, 1985.

[10] Pai R., Karmarkar, N., Rao, S.S.S.P., "A global router based on Karmarkar's Interior point method", CSE report, Indian Institute of Technology, Bombay, April 88.

[11] Karnath, A., Karmarkar N., Ramakrishnam, K. G., Resende M.G.C. "Computational Experience with an Interior-point Approach to the Satisfiability Problem" (in preparation).

[12] Vaidya, P., "A New Polynomial-time Algorithm for Minimization of Convex Functions Over Convex Sets" (in preparation).

Flow in Planar Graphs with Vertex Capacities

Samir Khuller *
Computer Science Department
Cornell University
Ithaca, NY 14853

Joseph Naor †
Computer Science Department
Stanford University
Stanford, CA 94305-2140

Abstract

Max-flow in planar graphs has always been studied with the assumption that there are capacities only on the edges. Here we consider a more general version of the problem when the vertices as well as edges have capacity constraints. In the context of general graphs considering only edge capacities is not restrictive, since one can reduce the vertex capacity problem to the edge capacity problem. However, in the case of planar graphs this reduction does not maintain *planarity* and cannot be used. We study different versions of the planar flow problem (all of which have been extensively investigated in the context of edge capacities).

1. Introduction

The computation of a maximum flow in a graph has been an important and well studied problem, both in the fields of Computer Science and Operations Research. Many efficient algorithms have been developed to solve the problem, e.g., [GT], [CH]. Research on flow in planar graphs is motivated by the fact that more efficient algorithms, both sequential and parallel, can be developed by exploiting the planar structure of the graph. This is important, in particular for parallel algorithms because maximum flow in general graphs was shown to be P-complete [GSS]. Planar networks arise in many contexts such as VLSI design and communication networks, and it is of interest to find fast flow algorithms for this class of graphs.

In the popular formulation of the planar flow problem one considers a single source vertex s and a sink t. Each edge has a capacity, and one wishes to find the max-flow from s to t. This problem has been extensively investigated by many researchers starting from the work by Ford and Fulkerson [FF] who developed an $O(n^2)$ algorithm for the special case of st-graphs (defined later). Subsequently, Itai and Shiloach developed an algorithm to find a max flow in an undirected planar graph [IS], which was improved by Reif [Re] who gave an algorithm to find the value of the max-flow in $O(n \log n)$ time. Hassin and Johnson [HJ] completed the picture by giving an $O(n \log^2 n)$ algorithm to compute the flow function. The problem of finding a minimum cut in a

*Supported by an IBM Graduate Fellowship. Part of this research was done while this author was visiting the IBM T.J. Watson Research Center.

†Supported by contract ONR N00014-88-K-0166 and by a grant of Stanford's Center for Integrated Systems.

planar directed graph was solved by [Jo] (both sequentially and in NC) who also gave algorithms to compute the flow function. Recently, Miller and Naor [MN] have pointed out that the general maximum flow problem in planar graphs is when there are many sources and sinks. They showed that when demands are fixed, the problem can be reduced to a circulation problem (with lower bounds on edge capacities). Note that here one cannot reduce the multiple source-sink problem to the single source-sink version since the reduction may destroy planarity. To summarize, if a planar graph is directed, then the complexity of computing the flow function is $O(n^{1.5})$ (assuming that the flow value has been given, which costs $O(n^{1.5}\log n)$ to compute).

In this paper we consider the version of the problem in which the vertices as well as edges have capacity constraints. Vertex capacities may arise in various contexts such as computing vertex disjoint paths in graphs and in various network situations when the vertices denote switches and have an upperbound on their capacities. For the case of general graphs this problem can be reduced to the version with only edges having capacity constraints by a simple idea of "splitting" vertices into two and forcing all the flow to pass through a "bottleneck" edge in-between. In planar graphs, this reduction may *destroy* the planarity of the graph and thus cannot be used. We show how to exploit the structure of the planar graph to develop efficient algorithms for the problem.

An application where vertex capacities play an important role is in reconfiguring VLSI/WSI arrays. Assume the processors on a wafer are configured in the form of a grid and due to yield problems, some are going to be faulty. Instead of treating the whole wafer as defective, the non-faulty processors can be reconfigured in the form of a grid. We assume that multiple data tracks are allowed along every grid line. It was shown in [RBK] that in this context, the reconfiguration problem can be abstracted combinatorially as finding a set of vertex disjoint paths from the faulty processors (the sources) to the boundary of the grid (the sink). The reader is referred to [RBK] and [RB] for more details and bibliography of this problem. (The main concern of [RB] is the single-track model).

We develop algorithms for computing the minimum cut when the graph has vertex and edge capacities. When the graph has only edge capacities the minimum cut corresponds to a cycle in the dual graph that separates the source from the sink. With vertex capacities the minimum cut consists of both edges and vertices. We show that for the single source-sink case, the minimum cut corresponds to a "cycle" in the dual graph when "jumping" over faces is permitted. Our algorithms are as fast as the corresponding algorithms for computing the minimum cut with only edge capacities. For the case of st-graphs we obtain an $O(n\sqrt{\log n})$ algorithm for finding the minimum cut (flow value). For the case of finding an $s-t$ minimum cut in an undirected planar graph, we are able to extend the algorithm by [Re], to obtain an $O(n\log n)$ algorithm for finding the value of the max-flow even when the graph has vertex capacities. To find the minimum cut in a directed planar graph is more expensive and costs $O(n^{1.5}\log n)$ [Jo] [MN]. Our algorithms also parallelize, yielding efficient NC algorithms.

For the case of st-graphs we obtain an $O(n\log n)$ algorithm to compute the flow function. The multiple source-sink problem with given demands, reduces to the circulation problem by a modified reduction of [MN]. To obtain a circulation we use the planar separator theorem to develop a divide and conquer algorithm that utilizes the st-graph case as a subroutine. The complexity of the algorithm is $O(n^{1.5}\log n)$.

2. Terminology and preliminaries

For each edge $e \in E$, let $D(e)$ be the corresponding *dual edge* connecting the two faces bordering e. Let $\mathcal{D} = (F, D(E))$ be the *dual graph* of G, where F is the set of faces of G and $D(E) = \{D(e) | e \in E\}$. There is a 1-1 correspondence between primal and dual edges and the direction of a primal edge e induces a direction on $D(e)$. We use a left hand rule: if the thumb points in the direction of e, then the index finger points in the direction of $D(e)$.

Associate with each edge $e \in E$ a capacity $c(e) \geq 0$, and also with each vertex $v \in V$ a capacity $c(v) \geq 0$. Let $S = s_1, \ldots, s_l$ and $T = t_1, \ldots, t_k$ be two sets of distinguished vertices, called *sources* and *sinks* respectively. A function $f : E \rightarrow Z$ is a legal flow function if and only if:

(i) $\forall e \in E : 0 \leq f(e) \leq c(e)$.

(ii) $\forall v \in V - \{S,T\} : \sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e)$

(iii) $\forall v \in V - \{S,T\} : \sum_{e \in in(v)} f(e) (= \sum_{e \in out(v)} f(e)) \leq c(v)$

Assume that G is biconnected, add edges with zero capacities appropriately to ensure that. Assume also that the vertices in S and T have no capacities. Otherwise, suppose that vertex $s \in S$ has capacity $c(s)$; add to the graph a new distinguished vertex $s' \in S$ adjacent only to s, such that the capacity of the edge joining s and s' is unbounded. Also remove vertex s from S. By performing this step for every capacitated vertex in S and T we obtain the required property.

The cost of a dual edge is defined to be the capacity of the corresponding primal edge.

In the maximum flow problem, we are looking for a legal flow function that maximizes the amount of flow entering T (or leaving S). The amount of flow entering the sink is also called the *value* of the flow function. A *circulation* is a legal flow function where condition (ii) is applied to every vertex in the graph, i.e., there are no sinks and sources.

A natural generalization of the flow problem is when edges have a lower bound different from zero on their capacity; in this case, both the lower bound and the upper bound may be either negative or positive and the capacity of an edge in that case will be denoted by $[a, b]$, where $a \leq b$.

We have the following equivalence rules that connect the orientation of an edge $e = (v \rightarrow w)$, the sign of its flow $f(e)$, and the sign of the lower and upper bounds on its capacity.

1. The edge $v \rightarrow w$ with flow $f(e)$ is equivalent to the edge $v \leftarrow w$ with flow $-f(e)$.
2. The edge $v \rightarrow w$ with capacity $[a, b]$ is equivalent to the edge $v \leftarrow w$ with capacity $[-b, -a]$.
3. The edge $v \rightarrow w$ with capacity $[a, b]$ is equivalent to two parallel edges: $v \rightarrow w$ of capacity b and $w \rightarrow v$ with capacity $-a$.
4. Let e_1 and e_2 be two parallel edges that are oriented in the same direction with capacities $[a_1, b_1]$ and $[a_2, b_2]$ respectively. The two edges can be replaced by one edge with capacity $[a_1 + a_2, b_1 + b_2]$.

Now, the definition of vertex capacities has to be changed slightly. Since the incoming flow into a vertex is equal to its outgoing flow, we require that, $\forall v \in V - \{S,T\} : \sum_{v \in e} |f(e)| \leq 2 \cdot c(v)$.

Miller and Naor [MN] reduced the problem of finding a flow in a graph with multiple sources and sinks (with specified demands), to the problem of computing a planar circulation with lower

bounds on the edges. We will concentrate on the circulation problem too, since their reduction can be modified appropriately to work even when vertices have capacities.

In the paper, the capacity of an edge may sometimes be also referred to as its *cost.*

The *residual graph* is defined with respect to a given flow. Let $e = (v, w)$ be an edge with capacity $[a, b]$ and flow f. In the residual graph, e is replaced by two darts with capacities $[0, b - f]$ and $[0, f - a]$.

A *spurious cycle* is a directed cycle along which the flow can be reduced, without any of the edges violating the lower bounds on their capacities.

A special case of planar flow is when the source and sink are on the same face. These graphs are called *st-planar graphs.*

A *potential* function $p : F \rightarrow Z$ is defined on the faces of a planar graph. Let e be an edge in the graph G, and let $D(e) = (g, h)$ be its corresponding edge in the dual graph such that $D(e)$ is directed from g to h. The potential difference over e is defined to be $p(h) - p(g)$. The following proposition, proved in [Ha] and [Jo], can be easily verified.

Proposition 2.1: *Let $C = c_1, \ldots, c_k$ be a cycle in the dual graph and let $f_1, \ldots, f_k$ be the potential differences over the cycle edges. Then, $\sum_{i=1}^{k} f_i(e) = 0$.*

It follows from the proposition that the sum of the potential differences over all the edges adjacent to a primal vertex is zero.

A potential function is defined to be *edge consistent* if the potential difference over each edge is not larger than its capacity. Such a potential function induces a circulation in the graph. If the circulation satisfies the vertex capacities as well, the potential function is defined to be *consistent.* The use of a potential function as a means of computing a flow was first suggested by Hassin [Ha], and was later elaborated by [HJ] and [Jo]. Miller and Naor too, use the idea of potentials to solve the problem of computing the circulation.

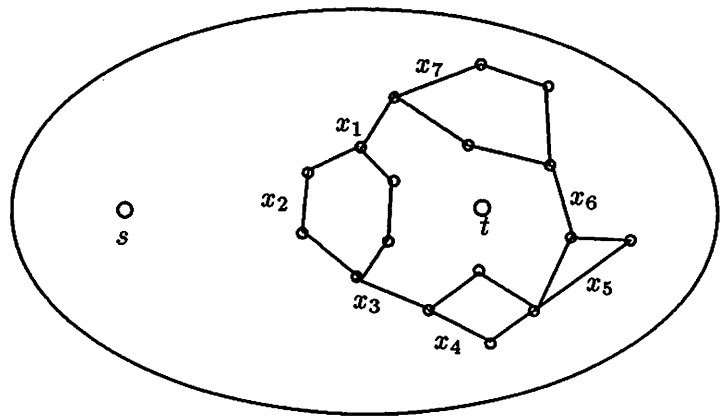
3. Computing the minimum cut

In this section, we show how to compute the minimum cut when vertex constraints are introduced into a graph containing a single source and sink. The problem of computing more efficiently (than in a general graph) the minimum cut when there are many sources and sinks is still open even when there are only edge capacities.

When the graph has vertex as well as edge capacities, the minimum cut is not just a set of edges; but a subset $S \subseteq E \cup V$ with the property that every path from s to t contains an element of S. Clearly, in the dual graph the minimum cut corresponds to a set of edges and a set of faces (that correspond to the vertices in the minimum cut). These edges and faces are "linked" together (see Fig. 1) and induce the shortest such cycle in the dual graph. We show how to modify the dual graph so that such a "linked" cycle can be computed.

In the dual graph we define a new shortest path computation as follows:

Definition 1: *Given a planar dual graph $\mathcal{D}$ with a cost c_{e_i} on each edge e_i, and a cost of c_{f_j} on each face f_j (this cost is the capacity of the corresponding primal vertex). We define a cycle to be a sequence of edges and faces $[x_1; x_2; \ldots; x_k]$ so that each x_i and x_{i+1} share a common vertex. (See Fig. 1 for an example.) The length of a cycle is the sum of the costs of the edges and the costs of*



$C = [x_1, x_2, ..., x_7]$

x_2, x_4, x_5, x_7 are faces in the dual graph

x_1, x_3, x_6 are edges in the dual

Figure 1: Cycle in the Dual Graph

the faces the cycle "jumps" over (to move from one edge to another). The shortest cycle is defined to be the cycle with the least length.

We show that the problem of computing a shortest cycle with jumping over faces, can be reduced to a shortest cycle problem in a planar graph with no jumping over faces.

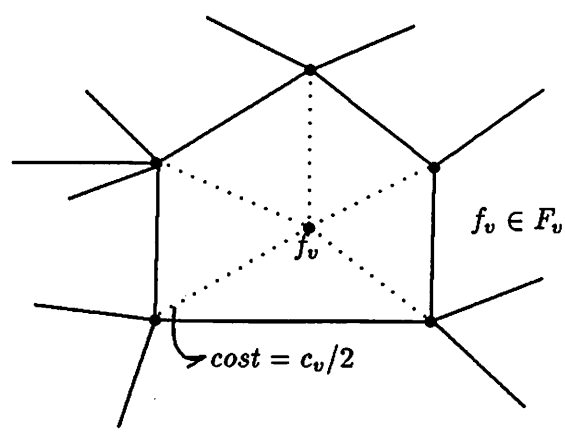


Figure 2: A face in the Dual Graph

Reduction: Given a planar dual graph $\mathcal{D}(F, D(E))$ with costs on the edges and faces, we compute a new planar dual graph $\mathcal{D}'(F', D(E)')$ as follows: Let $F' = F \cup F_V$, where each vertex in F_V corresponds to a face of $\mathcal{D}$ (that corresponds to a vertex in G). (Essentially for each face in $\mathcal{D}$ we introduce a new vertex in $\mathcal{D}'$.) Introduce edges in $\mathcal{D}'$ from f_v to every vertex on the corresponding face. Each of these edges is given a cost of $c_v/2$ where c_v is the capacity of the corresponding

primal vertex v. Thus $D(E)' = D(E) \cup \{(f_v, f_i) \mid f_i \in F\}$ (see Fig. 2). Clearly, $\mathcal{D}'$ is planar since f_v can be introduced in the corresponding face of $\mathcal{D}$. The cost of the minimum length path between two vertices incident on a face of $\mathcal{D}$ is clearly $\leq c_v$ (by going through f_v). The primal graph corresponding to $\mathcal{D}'$ will be denoted by G'.

Lemma 3.1: *The minimum cut in G is given by the length of the shortest cycle in $\mathcal{D}'$ that separates s from t.*

Proof: We show that a cycle in $\mathcal{D}'$ corresponds to a cut (with vertices and edges) in G. A cut that is an $s-t$ cut corresponds to a cycle that separates s from t. Clearly, the minimum cut corresponds to such a cycle of least length.

Consider a cycle C' in $\mathcal{D}'$. It is easy to see that C' in $\mathcal{D}'$ corresponds to a "linked" cycle C in $\mathcal{D}$ (of edges and faces). The vertices (edges) in C' that are not vertices (edges) in $\mathcal{D}$ correspond to the faces in $\mathcal{D}$ that are in C. The edges of cycle C correspond to edges of G that are in the cut, and the faces of C correspond to the vertices of G that are in the cut. Clearly, the faces of $\mathcal{D}$ (vertices of G) in the interior of C, are separated from the faces in the exterior of C. Thus the cycle C corresponds to a cut that separates s from t. □

We have now reduced the problem of finding the minimum cut in a planar graph with vertex capacities to that of finding the minimum cut in a new planar graph, G', that has only edge capacities.

The efficiency of computing the minimum cut in a planar graph varies with respect to whether the source and sink are on the same face, or whether the graph is directed. In the following subsections we handle the different cases. In the case of an st-graph (Section 3.1) and an undirected graph (Section 3.2), the known algorithms in the literature can handle the computation in the reduced graph $\mathcal{D}'$. If the graph is directed (Section 3.3), then the known method for computing minimum cut [MN] has to be modified.

3.1. The case of st-graphs

We shall begin by concentrating on the special case of st-graphs, when both the source and the sink are on the same face. We will assume this face to be the infinite face. Essentially we introduce two dual vertices s^* and t^* corresponding to the infinite face and construct the modified dual graph $\mathcal{D}'$ as defined earlier. The value of the maximum flow is given by the length of the shortest path from s^* to t^* (also called $u(t^*)$). This corresponds precisely to the *minimum cut* between s and t. Using Frederickson's algorithm [Fr], the shortest path from s^* to t^* can be found in $O(n\sqrt{\log n})$ time sequentially. To implement the same algorithm in parallel we use the algorithm by [PR] for computing shortest paths in planar graphs whose running time was improved by [GMN]. The algorithm runs in $O(\log^2 n \log\log n)$ time and uses $O(n^{1.5})$ processors on the EREW PRAM model.

Theorem 3.2: *We can compute the* value *of the max-flow in a directed st-planar graph in $O(n\sqrt{\log n})$ time. Moreover we can implement this algorithm in $O(\log^2 n \log\log n)$ time using $O(n^{1.5})$ processors on an EREW PRAM.*

3.2. The single source-sink case for undirected graphs

In [Re] it was shown that the minimum cut (or the value of the max flow) can be computed efficiently even when the vertices s and t are not on the same face in an undirected planar graph.

Using Frederickson's algorithm for shortest paths in planar graphs we obtain a running time of $O(n \log n)$. We note that combining the "jumping" over faces idea, along with Reif's algorithm, we get an $O(n \log n)$ time algorithm for computing the minimum cut in the graph even when vertices have capacities.

3.3. Matrix method to compute the minimum cut in a directed graph

The problem of finding a minimum cut (between a single source-sink pair) in the directed graph case is considerably harder and was dealt with by [Jo]. Recently, an elegant technique was developed by [MN] to find the minimum cut We show that an appropriate modification of the method is able to find the cut even when vertex capacities are present. The algorithm of [Jo] can be applied directly to G'.

Assume that the reduced graph $\mathcal{D}'$ has already been computed. The idea is to first "guess" the initial value f of the flow (can be taken to be the sum of the edge capacities leaving s). Then, we put a directed path p, from t to s to "return" the flow of value f, such that every edge on this path has capacity $[f, f]$. We would like to test whether there exists a circulation in the new graph. It is important that the edges of p are put in such a manner that no edge with a lower bound (on the return path) ever goes through a vertex that has a capacity. This can be done by adding new vertices. In Section 4.2 we discuss the more general problem of reducing a flow problem to a circulation problem; the reader is referred to that section for details on how the return path is inserted. Let the graph with the return path be denoted by G'' and its dual graph by $\mathcal{D}''$. Obviously, computing a circulation in G'' will induce a flow of value f in G (by deletion of the return path of flow f).

Following [MN], we have the following lemma.

Lemma 3.3: *If the dual graph $\mathcal{D}''$ contains a negative cycle, then there is no feasible circulation in G'', and we "guessed" too high a value for f.*

Proof: Let C be a cycle separating s from t in $\mathcal{D}'$. By introducing a return path p of capacity $[f, f]$, we would like to reduce the capacity of every such cycle C by f units. But, this will only happen if p does not meet C in a capacitated vertex; otherwise, the "jumping over faces" may ignore the negative capacity introduced by p. □

Our aim is to obtain the largest value of f so that $\mathcal{D}''$ does not have any negative cycles. In [MN], this value of f is found via a parametric search that takes at most $\log n$ iterations, where at each iteration f is updated, and the transitive closure is recomputed to check for negative cycles. To prove the correctness of this algorithm (Lemmas 6.1 and 6.2 in [MN]), two conditions must be met:

- All the negative edges have the same value, i.e., $-f$.
- All the negative edges form a directed path from t to s.

Since these conditions are satisfied in G'', we can apply this parametric search. [GMN] show how to implement the parametric search using the method of nested dissection of [LRT]. We obtain:

Theorem 3.4: *The minimum cut in a directed planar graph can be found in $O(n^{1.5} \log n)$. A parallel implementation uses $O(n^{1.5})$ processors and $O(\log^3 n \log \log n)$ time.*

4. Computing the flow function

In this section we assume that our planar graph contains many sources and sinks where the demand of each source and sink is known. We show how to compute a flow function that will satisfy the demands, or determine that a feasible circulation does not exist. The main difficulty in computing the flow function with vertex capacities is that the potential function computed in the dual graph with "jumping over faces" is not *consistent*. Consequently, computing the flow function becomes much more complicated than in the case where there are only edge capacities.

The first case we deal with is an st-graph. We present in Section 4.1 an $O(n \log n)$ implementation of the "uppermost path" algorithm due to Ford and Fulkerson [FF] that handles vertex capacities as well. In Section 4.2 we give a parallel algorithm to find the max-flow in an st-graph that works by cancelling the spurious cycles in the graph.

If the source and sink are not on the same face, or there are many sources and sinks in the graph, then the approach we take is to reduce the problem to that of computing a circulation. This is done in Section 4.3 similarly to [MN] by "piping" the flow back from the sinks to the sources, via a path that must not go through any capacitated vertices (this may involve addition of new vertices to G).

In Section 4.4 we present an efficient algorithm for computing a circulation when edges have lower bounds and vertices are capacitated.

4.1. The case of st-graphs revisited

Hassin [Ha] showed that it is possible to compute a maximum flow function (for the edge capacity case) as follows: for each edge $(i,j) \in E$, let $(i',j') \in D(E)$ be the associated dual edge. Then $f(i,j) = u(j') - u(i')$, where $u(j')$ $(u(i'))$ is the potential of the face (shortest distance to the vertex s^* in the corresponding dual graph). It is easy to see that this yields an edge consistent potential function and hence, a valid flow function.

In the case of vertex capacities, defining the flow through an edge to be the difference of the potentials of the faces on each side as computed in $\mathcal{D}'$ (with jumping over faces), yields a flow function that may violate vertex capacity constraints. We illustrate the inconsistency of the potential function by an example in Fig. 3. All the capacities of the edges that are not shown are considered to be high. The min-cut is due to the edge incident at t of capacity 1, and the vertex v of capacity 2. The rest of the potentials are as shown on the faces of the graph. Note that v is the vertex that has excess flow through it. The cancellation of the "spurious cycle" of length three $(v - v_1 - v_2 - v)$, of unit flow, rooted at v has the effect of producing a valid flow function, without changing the value of the max-flow.

We now give an $O(n \log n)$ algorithm to compute a valid flow function in an st-graph that has vertex capacities. For details on the uppermost path algorithm we refer the reader to [IS] and [FF] (see also [NC]). An efficient $O(n \log n)$ implementation of this algorithm was given by [IS]. We will describe the algorithm for undirected graphs, the algorithm easily extends to directed st-graphs as well.

We will briefly outline the modifications to the algorithm to handle vertex capacities. The algorithm begins by pushing flow through the uppermost path from s to t (see Fig. 4).

The capacity of the uppermost path is defined to be the least residual capacity of either an edge or a vertex. At least one edge, or vertex on the uppermost path gets saturated by pushing a flow of value equal to the capacity of the path. The saturated edge/vertex is deleted from the graph,

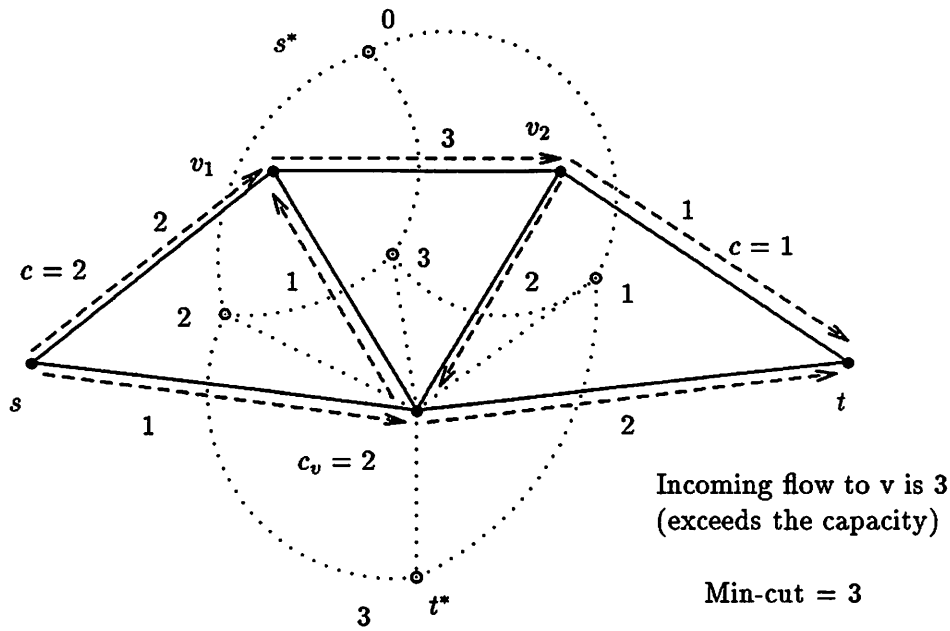


Figure 3: Example to show violation of vertex capacities

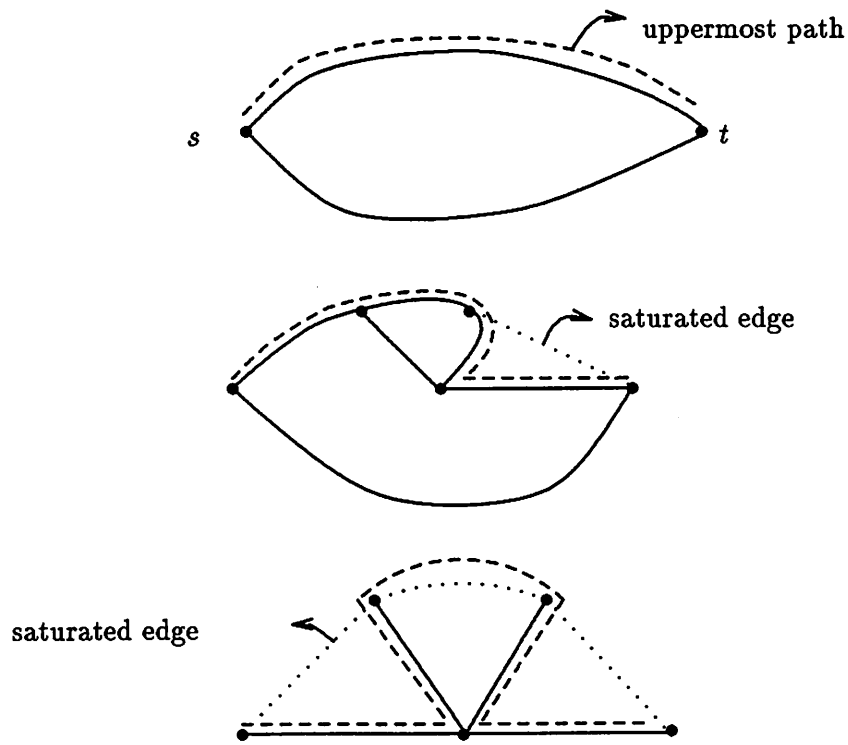


Figure 4: Uppermost path may be non-simple

and the process is repeated using the uppermost path in the residual graph until s is disconnected from t.

Care needs to be taken to make the uppermost path *simple* each time we delete the saturated edge or vertex. The reason for this is the presence of vertex capacities in the graph. In the case of only edge capacities, pushing a flow of value equal to the capacity of the uppermost path does not violate any capacity constraint. Suppose there are vertex capacities present and the path is non-simple at a vertex that has a capacity. Now pushing a flow of value f on this path actually increases the incoming flow to this vertex by atleast $2f$ units – which could cause a violation in the capacity constraint of this vertex. This is the main modification to the algorithm presented in [IS]. As a result the algorithm discards pieces of the graph in making the path simple at each pushing step. By the following proposition we can show that the value of the flow function computed by this modified uppermost path algorithm is the same as the value of the min-cut.

Proposition 4.1: *The process of making the augmenting path simple at each step does not decrease the amount of flow pushed on that augmenting path.*

The capacities of the edges and vertices are kept in two separate heaps. At each step we choose the minimum from each heap, which defines the amount of flow to be pushed at that step. After deleting the appropriate edge or vertex we need to update the uppermost path to make it simple. If the bottleneck was due to an edge, this is done by simply traversing the edges on the face adjacent to the edge (the face other than the external face) and introducing them into the heap. During this traversal, we can detect if any of the vertices already belong to the uppermost path and are causing a non-simplicity. If this condition is detected the entire subchain causing the non-simplicity is discarded and the corresponding values from the heap are deleted. (See Fig. 4). However, note that this is done only once for each edge/vertex (when they are introduced into the uppermost path), after they are deleted they stay deleted and are never encountered again. If the bottleneck on the path is due to a vertex v, to recompute the new uppermost path we need to traverse the faces adjacent to v (clockwise from the forward edge of the path incident on v). Again during the traversal, we ensure that the new path is simple and discard all the edges and vertices that belong to the segment of the path causing the non-simplicity.

Theorem 4.2: *A maximum flow function can be computed in $O(n \log n)$ time for the case of directed st-planar graphs, even when the vertices have capacities.*

4.2. The parallel algorithm

We develop a two phase parallel algorithm to find a valid max-flow in the case of st-planar graphs. In the first phase we compute potentials by a shortest path computation from s^* (with jumping over faces permitted). If there are no capacitated vertices then clearly this yields a valid flow function. In certain cases, it may also happen that this procedure yields a valid flow function even in the presence of vertex capacities. In general, it does not yield a valid flow function (as the earlier example showed) due to the presence of "spurious cycles".

In the second phase we show how to fix all the "unhappy vertices" (that have excess flow through them). To motivate the second phase let us see what goes "wrong" when we compute potentials via jumping over faces. The uppermost path algorithm really corresponds to growing a shortest path tree from s^*, where the augmenting path at each step directly corresponds to the "fringe" of the dual tree at various stages of a Dijkstra shortest path computation. When the fringe of the

tree is non-simple then the uppermost path is non-simple and needs to be made simple. The flow function computed by assigning potentials, directly corresponds to an uppermost path algorithm without making the path simple at each step – this is precisely what causes excess flow to go through capacitated vertices. In the second phase we will try and cancel all the "spurious cycles" that cause capacitated vertices to be unhappy.

Let the region R^1 be initially defined to be $f(s^*)$. (Where $f(v)$ denotes the face corresponding to dual vertex v.) We are going to assume that the infinite face has been divided into two faces (separating s^* from t^*) by the addition of a return edge from t to s of infinite capacity.

In general, R is the set of faces that have been "reached" by the dual shortest path tree. Each time an edge on the uppermost path is saturated, it corresponds precisely to growing the region R by annexing a new face (the next "closest" face from s^*). Each uppermost (augmenting) path corresponds directly to a fringe of the dual tree at a snapshot of the algorithm. We would like to identify the various augmenting paths and make them simple at each step by completely discarding the "loops" as was done in the uppermost path algorithm.

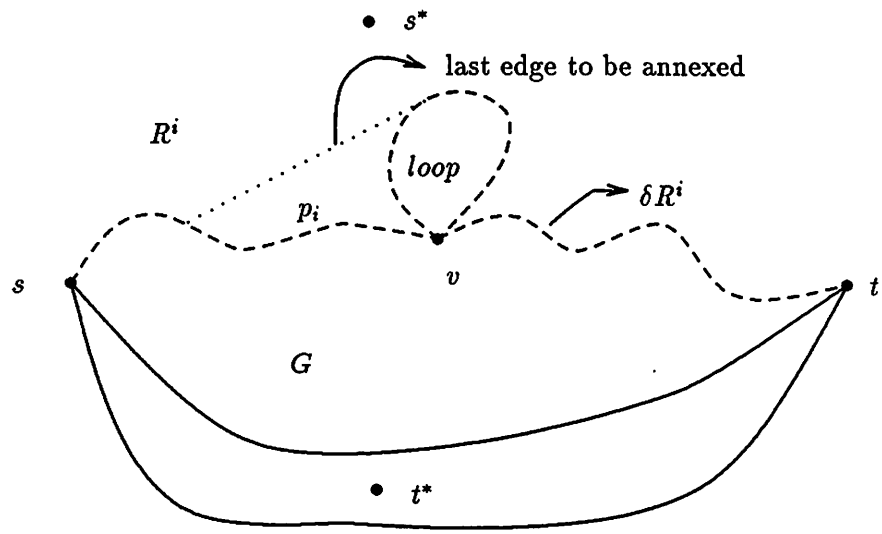


Figure 5: Region R^i, the dual tree T_D^i and a "loop"

In the first phase we computed the dual tree T_D. In the second phase we consider various sub-tree's as follows:

$$T_D^i = \{v \mid p(v) \leq p_i\}$$

Assuming all the potentials have been sorted in non-decreasing order $p_1, p_2, \ldots, p_f$, where f is the number of faces.

We will also assume that the embedding of each T_D^i is also known (this can be obtained from the embedding of G and its dual). We define R^i to be the union of the set of faces in T_D^i, and δR^i to be the boundary of R^i. Let the fringe of T_D^i be δR^i.

We are going to lower the potentials of certain faces, and then define the flow to be the difference of potentials of the adjacent faces. As before, the convention is that the flow moves in the direction with the smaller potential on the left.

In parallel, we need to identify the non-simplicities in δR^i. In Fig. 5 we show the formation of a non-simplicity at a capacitated vertex and how to get rid of it. Suppose that the non-simplicity is formed in T_D^i, and is not present in T_D^{i-1}, then we define the "loop" to be the finite region enclosed by the non-simple portion of δR^i (see Fig. 5).

We now redefine the potential of all the faces in the "loop" to be p_i. This has the effect of making the flow on *all* the internal edges of the loop to be zero. This ofcourse has the effect of making all the unhappy vertices in the interior of the loop to be happy (and we instantly inform the processors that are trying to fix these unhappy vertices that the vertices are now fixed).

Lemma 4.3: *Setting the potentials of all the faces in the loop in T_D^i to be p_i, preserves edge capacities.*

Proof: The flow on all the internal edges of the loop becomes zero since the potentials of both the adjacent faces is p_i. Notice that the flow on the edges of the boundary of the loop is only decreased (since the higher potentials (of faces in the loop) are lowered, this keeps the flow to be less than the capacity of the corresponding edge). □

This has the effect of "deleting" the loop and making the augmenting path simple. In this manner we can identify all spurious cycles and cancel them, obtaining a flow function that does not violate any capacitated vertices.

The loops are cancelled in the first dual tree that they occur (i.e., a loop is cancelled in T_D^i, if it is not contained in a loop in T_D^j for $j < i$). After all the potentials have been adjusted in this manner we can show the following property:

Lemma 4.4: *For a capacitated vertex v, if we consider the sequence of potentials of faces incident on it (in order) starting from the smallest potential, there is only one distinct local maxima.*

Proof: If v belongs to the interior of some loop, then the property is trivially true as all the potentials of the faces incident on it are the same. Suppose that there are two locally maximal faces (the proof is similar for more than two such faces). Let the potentials of the two faces be p_{v_1} and p_{v_2}. The faces in-between these two faces (from both sides) all have lower potential. Clearly, in some dual tree there will be a non-simplicity formed in the boundary that includes either p_{v_1} or p_{v_2} in the loop, which would have caused a lowering of one of p_{v_1} or p_{v_2}. □

Using this property it is easy to see that all the capacitated vertices do not have any excess flow through them. (Since the maximum difference of potentials of any two faces incident to v is bounded by c_v.) We can now bound the incoming and outgoing flows by c_v.

By using the fast matrix multiplication of [PR] and its improvement by [GMN] we obtain:

Theorem 4.5: *A max-flow in an st-graph can be found in $O(\log^2 n \log\log n)$ time using $O(n^2)$ processors.*

Proof: We use the algorithm by [GMN] to compute the shortest path tree in parallel (tree rooted at s^*). The second phase is easy to implement in $O\log n)$ time using $O(n)$ processors for each dual tree T_D^i. □

4.3. Reduction from flow to circulations

We now show how to transform a flow problem to a circulation problem with lower bounds on the edges. This will be done by adding new edges that will return the flow from the sinks back to the sources. These edges will have lower bounds so as to insure that the demands of the sources and sinks are satisfied.

We first compute a spanning tree T in G. (We treat G as an undirected graph for the purpose of computing T.) Notice that if there is only one source and sink, then T is a path. Let us denote the demand of a vertex v by $d(v)$ and assume that if v is a source, then $d(v) < 0$, whereas if v is a sink, then $d(v) > 0$.

An edge $e \in T$ separates the tree into two subtrees, called tail and head. T_{tail} is the subtree adjacent to the tail of e and T_{head} adjacent to the head of e. Let $r(e)$ be $\sum_{v \in T_{tail}} d(v) = -\sum_{v \in T_{head}} d(v)$. The previous equality follows from the fact that T is a spanning tree and the sum of the demands is zero. We will add an edge parallel to e (with the same direction) whose capacity is $[r(e), r(e)]$.

In the new graph we will compute a circulation and obtain a legal flow that satisfies the demands by removing the newly added tree edges.

Let us now elaborate on how the return flow edges are inserted without being adjacent to capacitated vertices. Assume that T is a simple path. The procedure easily generalizes to trees. T is inserted edge-by-edge; assume that we are currently inserting the edge $v \rightarrow w$ with return flow $[l, l]$, where w is capacitated and v is not. This condition is initially satisfied as the sources and sinks are not capacitated (see Fig. 6).

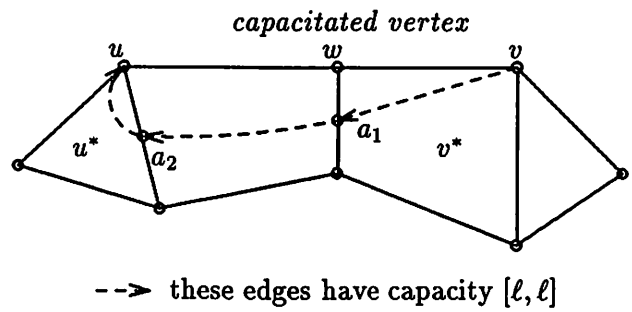


Figure 6: Embedding the edge $v \rightarrow w$

Let u be the vertex that succeeds w on the path and let u^* and v^* be faces adjacent to u and v respectively. Let $e_1^*, \ldots, e_k^*$ be a dual path from v^* to u^* and let $e_1, \ldots, e_k$ be the corresponding primal edges. The idea is that the path will arrive at u through the edges $e_1, \ldots, e_k$ instead of w. To do that, we split each edge e_i into two edges, e_i' and e_i'', by adding a vertex a_i in the middle. The path from v to u will go through the vertices $a_1, \ldots, a_k$ by adding new edges with capacity $[l, l]$. This is repeated for each edge on T and notice that the last vertex on T is not capacitated. In this manner we are able to "bypass" all the capacitated vertices to return the flow from the sinks to the sources.

Since the degree of each vertex a_i is exactly two, by conservation of flow the flow on e_i' and e_i'' is the same, both in value and direction. Hence, when the return edges and the vertices a_i are removed, we still have a legal flow.

This procedure increases the size of the graph by $O(n)$ vertices, since there are $O(n)$ edges of the spanning tree that need to be embedded.

4.4. Computing Circulations

In this section we show how to use the planar separator theorem [Mi] to obtain a solution for the circulation problem when the graph contains edge capacities (upper and lower) as well as vertex capacities. This approach is similar to the algorithm developed by [JV].

An overview of the algorithm:

Step 1. Find a separating cycle C of size $O(\sqrt{n})$. Let the *interior* and *exterior* of G be denoted by G_I and G_E.

Step 2. Recursively find a circulation in $G_I + C$ and $G_E + C$.

Step 3. Merge the circulations computed in Step 2, to obtain a circulation in G.

Let us now elaborate on each step. In the first step, we compute a separating cycle of size $O(\sqrt{n})$. However, we require a separating cycle that has no capacitated vertices on it. To obtain such a cycle, we first find a separating cycle in the dual graph. The vertices on this cycle correspond to a set of faces in the primal graph, such that the faces corresponding to the adjacent vertices on the cycle share a common edge (edges in E_C). This cycle corresponds to a set of faces in the primal graph that can be decomposed into two cycles C_1 and C_2 and a set of edges E_C between them. We introduce a new vertex in the middle of each edge in E_C, and connect a cycle C through the new vertices by adding new edges with zero capacity. (See Fig. 7). Call the new graph G'. This cycle

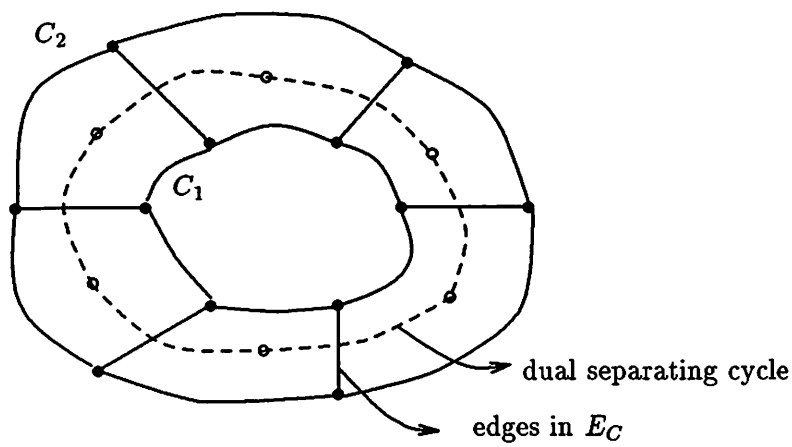


Figure 7: Finding an uncapacitated separating cycle

is directed clockwise, and its edges have zero capacity.

Lemma 4.6: *The cycle C is a separator of size $O(\sqrt{n})$ such that the number of vertices in G_I and G_E is bounded by cn ($c < 1$).*

Lemma 4.7: *Any circulation in G' induces a circulation in G.*

Proof: Since all the extra edges added in G' have zero capacity, a circulation in G' directly yields a circulation in G by dropping the edges that have zero capacity (and thus no flow). We can also easily drop the vertices that were added to create the separating cycle. □

In the second step, we compute a circulation recursively in $G_I + C$ and $G_E + C$. Since edges have lower bounds, we have to ensure that a feasible solution exists. This is done by giving each edge on C an infinite capacity.

Lemma 4.8: *There exists a feasible solution in $G_I + C$ and $G_E + C$.*

Proof: Let us assume that there is a feasible circulation C in G. We show that in $G_I + C$ there is also a feasible circulation after the capacities of the vertices have been changed to infinity. (The proof for $G_E + C$ is similar.) Consider the restriction of C to G_I. Clearly all the vertices strictly in the interior of C satisfy the flow conservation requirement (as they are not affected in any way). The only vertices that are affected are the ones on the boundary of the cycle C (the new uncapacitated vertices). Some of them suddenly become deficient in the flow they receive (due to deletion of the edges in the exterior of C) and some have excess flow. Since C was a legal circulation, the sum of the excesses equals the sum of the deficiencies. The edges of C are now used to redistribute all the excesses to the deficient vertices. Hence the graph $G_I + C$ has a legal circulation. □

In the third step, we merge the two solutions obtained previously. The merged flow satisfies the flow conservation constraint for each vertex, but the capacity of the edges of the separator is violated. Notice, that vertex capacities are not violated due to the merging step.

Let $e = (v \rightarrow w)$ be an edge of C, with flow f_e. Since it has zero capacity the flow from v to w needs to be redirected (i.e., v becomes a source that needs to send f_e units of flow to the sink w) in the residual graph. Notice that in the residual graph there are no lower bounds on edges and that in this sub-problem v and w are on the same face. A solution to this problem can be obtained by using a modification of the algorithm presented earlier for st-graphs. (In pushing flow in the residual graph, vertex capacities are "active" for only a subset of the edges, since when we push flow on an edge it is actually either increasing or decreasing the flow through the vertices incident on it.) It is easy to see that a simple modification to the algorithm presented in the earlier section will work. This procedure is repeated for each edge on the separator. Observe that if the vertices of C were *capacitated* then it is not possible to simply merge the solutions.

If at any step there is no way of redirecting the flow in the residual graph from v to w, then the algorithm halts and claims that there is no circulation in the graph. The correctness of this claim can be easily seen from the following argument. Suppose that there is a circulation f in the original graph, and let f' be the current circulation, in which the flow cannot be redirected from v to w. Then, $f - f'$ is a collection of residual cycles such that augmenting them in f' will redirect the flow from v to w.

Since the size of the separator is $O(\sqrt{n})$ we obtain:

Theorem 4.9: *The complexity of computing a circulation in a planar graph with vertex capacities is $O(n^{1.5} \log n)$.*

Proof: The algorithm makes recursive calls to two graphs that are bounded in size by $\frac{2}{3}n + O(\sqrt{n})$ (where n is the number of vertices in the graph). The cost of merging the solutions is $O(n^{1.5} \log n)$ as we make $O(\sqrt{n})$ calls to a procedure that pushes a given amount of flow in an st-planar graph. Each call to the algorithm costs us $O(n \log n)$ time (using the algorithm presented in the previous section).

□

Notice that our algorithm for computing a circulation is slower than that of [MN] (for edge capacities) by only a logarithmic factor. For the case of computing a max-flow the complexity of our algorithm is the same as that of computing a min-cut (the value of the max-flow).

5. Conclusions and open problems

We have shown a simple reduction for computing the minimum cut in a graph with capacitated vertices to a graph that has only edge capacities. However, this reduction holds only if there is one source and sink. If there are many sources and sinks, then it is not true anymore that the minimum cut is equal to a collection of cycles of minimum capacity that separates the sources from the sinks in $\mathcal{D}'$, i.e., with "jumping over faces". The reason is that two cycles in this collection are not necessarily "independent" (if they share a common capacitated vertex). We conjecture that if there many sources and sinks, then a simple reduction of the above form does not exist.

It seems that the major difficulty with vertex capacities is in computing the flow function. Suppose that we want to compute the flow function via a potential function in a similar way to [MN]. As already pointed out, even if we use "jumping over faces" for computing the potential function, we do not get a legal circulation necessarily. (See Fig. 3). To obtain a legal circulation, a set of spurious cycles has to be identified and cancelled. Can these cycles be efficiently identified? If the graph contains only one source and sink, then the spurious cycles have a more special structure. In every spurious cycle, the flow on an edge needs only to be decreased and never increased. Can the spurious cycles in this case be efficiently identified ? In the case of undirected graphs with a single source and sink, our algorithm is slower than that of [HJ]. We conjecture that the special structure of the spurious cycles will enable them to be cancelled easily.

In the case of *st*-planar graphs, the cycles have a special structure that is exploited by the parallel algorithm. We conjecture that an $O(n\sqrt{\log n})$ exists to compute the flow function for *st*-graphs that works by cancelling these spurious cycles.

Another natural open problem is how to compute the flow function in parallel. We can do that only for *st*-graphs. Can that be done for more general classes of planar graphs? How difficult is it to compute a circulation in parallel (with vertex capacities) ?

Acknowledgements

We would like to thank Donald Johnson, Gary Miller and Vijay Vazirani for fruitful discussions.

References

[CH] J.Cheriyan and T.Hagerup, *A randomized $O(nm + n^2(\log n)^4)$ maximum flow algorithm,* 30th Annual Symposium on Foundations of Computer Science (1989).

[FF] L. R. Ford and D. R. Fulkerson, *Maximal flow through a network,* Canadian Journal of Mathematics, Vol. 8, pp. 399-404 (1956).

[Fr] G. N. Frederickson, *Fast algorithms for shortest paths in planar graphs, with applications,* Siam Journal on Computing, Vol.16, pp. 1004-1022 (1987).

[GT] A.V.Goldberg and R.E.Tarjan, *A new approach to the maximum flow problem,* Journal of the ACM, vol 35, 4, pp. 921-940 (1988).

[GSS] L. Goldschlager, R. Shaw and J. Staples, *The maximum flow problem is log space complete for P,* Theoretical Computer Science, Vol. 21, pp. 105-111 (1982).

[Ha] R. Hassin, *Maximum flows in (s,t) planar networks,* Information Processing letters, Vol. 13, page 107 (1981).

[HJ] R. Hassin and D. B. Johnson, *An $O(n\log^2 n)$ algorithm for maximum flow in undirected planar networks*, Siam Journal on Computing, Vol. 14, pp. 612-624 (1985).

[GMN] H. Gazit, G. L. Miller and J. Naor, *A fast parametric search method in planar graphs*, manuscript.

[IS] A. Itai and Y. Shiloach, *Maximum flow in planar networks,* Siam Journal on Computing, Vol. 8, pp. 135-150 (1979).

[Jo] D. B. Johnson, *Parallel algorithms for minimum cuts and maximum flows in planar networks,* Journal of the ACM, Vol. 34, (4), pp. 950-967 (1987).

[JV] D. B. Johnson and S. Venkatesan, *Using divide and conquer to find flows in directed planar networks in $O(n^{1.5}\log n)$ time,* Proceedings of the 20th Annual Allerton Conference on Communication, Control and Computing, University of Illinois, Urbana-Champaign, Ill., pp. 898-905 (1982).

[LRT] R. J. Lipton, D. J. Rose and R. E. Tarjan, *Generalized nested dissection*, Siam J. Numerical Analysis, Vol. 16, pp. 346-358 (1979).

[Mi] G.L.Miller, *Finding small simple separators for 2-connected planar graphs*, Journal of Computer and System Sciences, 32 (1986), pp 265-279.

[MN] G. L. Miller and J. Naor, *Flow in planar graphs with multiple sources and sinks*, Proceedings of the 30th Annual Symposium on Foundations of Computer Science (1989), pp. 112-117.

[NC] Nishizeki and Chiba, *Planar Graphs: Theory and Algorithms*, Annals of Discrete Math (32), North Holland Mathematical Studies.

[PR] V. Pan and J.H. Reif, *Fast and efficient parallel solution of linear systems,* to appear, Siam Journal on Computing.

[Re] J. H. Reif, *Minimum $s-t$ cut of a planar undirected network in $O(n\log^2 n)$ time,* Siam Journal on Computing, Vol. 12, No. 1, pp. 71-81 (1983).

[RB] V. P. Roychowdhury and J. Bruck, *On finding non-intersecting paths in a grid and its application in reconfiguration of VLSI/WSI arrays,* to appear, Proceedings of the 1^{st} Symposium on Discrete Algorithms (1990).

[RBK] V. P. Roychowdhury, J. Bruck, and T. Kailath, *Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays*, to appear in IEEE Trans. on Computers: Special Issue on Fault Tolerant Computing, (1990).

On the Radon Number of the Integer Lattice

Shmuel Onn*†

February 5, 1990

Abstract

In this paper we investigate the Radon number $r(d)$ of a natural convexity space on the integer lattice, $\mathbf{Z^d}$. We give an $\Omega(2^d)$ lower bound and an $O(d2^d)$ upper bound, prove $r(2) = 6$, and establish the existence of Radon partitions in lattice polytopes having a large stable set of vertices. We discuss the computational complexity of deciding if a given set of points in $\mathbf{Z^d}$ has a Radon partition, and show that if d is fixed, then this problem is in P, while if d is part of the input, it is NP-complete.

Key words. Abstract Convexity, Convexity Spaces, Geometry of numbers, Radon Number, Radon Partition, Lattice Polytopes, Integer Programming, Integer Lattice.

1 Introduction

Let us begin with a few definitions.
A convexity space is a pair (X, C), where X is a set, $C \subset 2^X$, and the following hold:

1. $\phi \in C,\ X \in C$
2. $\forall F \subset C\ \ [\cap F \in C]$.

The C-hull of a set $A \subset X$ is

$$C(A) = \cap\{B : B \in C \ \wedge\ A \subset B\}.$$

The classic example is $(\mathbf{R^d}, conv)$, where

$$conv = \{A : A \subset \mathbf{R^d} \wedge A \text{ is convex}\}.$$

Fix a convexity space (X, C). For any $F \subset 2^X$, define the following.

Definition 1.1 *Caratheodory number.*
The Caratheodory number $c(F)$ is the infimum over all $n \in N$ such that the C-hull of any set $S \in F$ is the union of the C-hulls of subsets $T \subset S$ with $|T| \leq n$,

$$c(F) = \inf\{n \in N : \forall S \in F\ C(S) = \cup\{C(T) : T \subset S \ \wedge\ |T| \leq n\}\}.$$

Definition 1.2 *Helly number.*
The Helly number $h(F)$ is the infimum over all $n \in N$ such that any finite subfamily $L \subset F$ with the property

*Research partially supported by NSF grant DMS8706133 through the Center for Applied Mathematics, Cornell University.
†School of Operations Research and Industrial Engineering, Cornell University.

that the intersection of the C-hulls of all its members, $\cap\{C(A) : A \in L\}$, is empty, has a subfamily $M \subset L$ of size $|M| \leq n$ with the same property,

$$h(F) = \inf\{n \in N : \forall L \subset F \ [|L| < \infty \wedge \cap\{C(A) : A \in L\} = \phi] \rightarrow [\exists M \subset L \ |M| \leq n \wedge \cap\{C(A) : A \in M\} = \phi]\}.$$

Definition 1.3 *Radon number and Radon partition.*
The Radon number $r(F)$ is the infimum over all $n \in N$ such that any set $A \in F$ with $|A| \geq n$ has a partition (A_1, A_2), called "Radon partition", such that the C-hulls of the two disjoint sets A_1 and A_2 intersect,

$$r(F) = \inf\{n \in N : \forall A \in F \ |A| \geq n \rightarrow [\exists A_1 \exists A_2 \ A_1 \cup A_2 = A \wedge A_1 \cap A_2 = \phi \wedge C(A_1) \cap C(A_2) \neq \phi]\}.$$

Definition 1.4 *exchange number.*
The exchange number $e(F)$ is the infimum over all $n \in N$ such that for every point $p \in X$ and finite set $A \in F$, if $|A| \geq n$ then the C-hull of A is contained in the union of the C-hulls of sets obtained from A by replacing a point $a \in A$ by the point p,

$$e(F) = \inf\{n \in N : \forall p \in X \ \forall A \in F \ [n \leq |A| < \infty] \rightarrow [C(A) \subset \cup\{C(p \cup (A - a)) : a \in A\}]\}.$$

When the underlying set of the space is a cartesian product $X = K^d$, we will use $c(d, F)$, $h(d, F)$, $r(d, F)$, $e(d, F)$. For $F = 2^X$ we simply use c, h, r, e, and in the cartesian product case $c(d), h(d), r(d), e(d)$.
It is well known that for $(\mathbf{R^d}, conv)$,

$$c(d) = d + 1, \quad h(d) = d + 1, \quad r(d) = d + 2, \quad e(d) = d + 1.$$

Throughout the paper, by a "polytope" we mean a convex polytope, i.e., a set $P = conv(V) \subset \mathbf{R^d}$ such that V is finite. If $V \subset \mathbf{Z^d}$, then P is called a "lattice polytope".

There are several papers discussing convexity spaces and the Helly, Radon, Caratheodory and exchange numbers, e.g. [Danzer, Grunbaum and Klee], [Eckhoff], [Hammer], [Levi]. Recently, some investigations motivated by Integer Programming were concerned with Helly properties of the integer lattice (Bell, Scarf, see [Shrijver]), and Caratheodory properties of it ([Cook, Fonlupt and Shrijver]).

The purpose of this paper is to study the Radon number of the integer lattice, namely $r(d)$ for the family of convexity spaces $(\mathbf{Z^d}, C_d)$, where, for $A \subset \mathbf{Z^d}$, $C_d(A) = conv(A) \cap \mathbf{Z^d}$, and $C_d = \{C_d(A) : A \subset \mathbf{Z^d}\}$. This family of convexity spaces had been studied before in [Doignon].

In contrast with the linear growth rate, as a function of the dimension, of the Radon number of the reals given above, we show in Section 2 an $\Omega(2^d)$ lower bound on the Radon number $r(d)$ of the integer lattice. It is interesting to note that this lower bound is also in contrast with the linear growth rate of the Radon number of the product integral convexity space $(\mathbf{Z}, C_1)^d$, defined on the same ground set $\mathbf{Z^d}$, but having only the collection $\{A_1 \times A_2 \times \cdots \times A_d : A_i \subset \mathbf{Z}, \ 1 \leq i \leq d\}$ of d-boxes as its family of convex sets. The linear growth of the Radon number for that space follows from an upper bound on Radon numbers of product convexity spaces given in [Eckhoff].

In Section 3, we show an $O(d2^d)$ upper bound on the Radon number of the integer lattice. It is easy to see that, in the definition of the Radon number, it is enough to consider subsets $A \subset X$ such that $|A| < \infty$ and $A = ext(P)$, the set of vertices of P, where P is some lattice polytope. Thus, the question about $r(d)$ is in fact a question about the existence of Radon partitions of sets of vertices of lattice polytopes, and one can ask the same question about subclasses of lattice polytopes. This is the subject of Section 4, in which we study some properties of polytopes having a large stable set of vertices (in graph theoretical sense), and establish the existence of a Radon partition for such polytopes. This result is also useful for the proof, given in Section 5, that $r(2) = 6$, and leads to a simpler and more direct proof, for simple lattice polytopes, of the upper bound.

Finally, in Section 6, we discuss the computational complexity of deciding if a given set of points in $\mathbf{Z^d}$ has a Radon partition, and show that if d is fixed, than this problem is in P, while if d is part of the input, it is NP-complete.

Before going on, we note that the Radon number of the integer lattice is invariant under affine transformations. More precisely, we have:

Observation 1.5 Consider a set $S \subset \mathbf{Z^d}$ such that $dim(aff(S)) = k \leq d$. If $|S| \geq r(k)$, then S has a Radon partition.

The proof is based on the fact that, given such a set, there is an affine bijection from $aff(S)$ to $\mathbf{R^k}$ which preserves integrality (this is a result, say, of Thm.3 p. 19 in [Gruber and Lekkerkerker]).

2 A Lower Bound

Proposition 2.1 For $d \geq 1$ we have

$$r(d+1) \geq 2r(d) - 1.$$

Proof: Assume indirectly that $r(d+1) \leq 2r(d) - 2$. Let $A \subset \mathbf{Z^d}$, $|A| = r(d) - 1$ be a set that admits no Radon partition. Let

$$X_i = \{(a, i) : a \in A\} \quad (i = 0, 1).$$

Then $X_0 \cup X_1 \subset \mathbf{Z^{d+1}}$ and $|X_0 \cup X_1| = 2(r(d) - 1) \geq r(d+1)$, so there exists a Radon partition of $X_0 \cup X_1$ into two sets Y_0, Y_1. Let $p \in C_{d+1}(Y_0) \cap C_{d+1}(Y_1)$. Then $p \in \mathbf{Z^{d+1}}$, and $C_{d+1}(Y_0) \cap C_{d+1}(Y_1) \subset \{(x_1, \ldots, x_{d+1}) : 0 \leq x_{d+1} \leq 1\}$, so $p_{d+1} \in \{0, 1\}$. Assume, without loss of generality, that $p_{d+1} = 0$. Then, for $i = 0, 1$, let $Z_i = Y_i \cap \{(x_1, \ldots, x_{d+1}) : x_{d+1} = 0\}$. We then have that $p \in C_{d+1}(Z_0) \cap C_{d+1}(Z_1)$. Thus, the sets $\{a : (a, 0) \in Z_0\}$ and $\{a : (a, 0) \in Z_1\}$ form a Radon partition of A, a contradiction.

The following set of points (see figure 1) shows that $r(2) \geq 6$.

$$\{(0,0), (2,0), (0,1), (1,2), (3,2)\}.$$

This, together with Proposition 2.1, yield the following.

Corollary 2.2 For $d \geq 2$, $r(d) \geq 2^d + 2^{d-2} + 1 = 5 \cdot 2^{d-2} + 1$.

Proof: By induction on d. for $d = 2$ this is true since $r(2) \geq 6$. Now,

$$r(d+1) \geq 2r(d) - 1 \geq 2 \cdot (2^d + 2^{d-2} + 1) - 1 = 2^{d+1} + 2^{(d+1)-2} + 1,$$

as required.

3 An Upper Bound

The following is concerned with Helly properties of $(\mathbf{Z^d}, C_d)$.

Proposition 3.1 (Bell and Scarf, see [Shrijver], Thm. 16.5)
Let F be a finite collection of closed half spaces in $\mathbf{R^d}$. If $(\cap F) \cap \mathbf{Z^d} = \phi$ then there exists a subfamily $L \subset F$ such that $|L| \leq 2^d$ and $(\cap L) \cap \mathbf{Z^d} = \phi$.

Let $FIN = \{A \subset \mathbf{Z^d} : |A| < \infty\}$.

Corollary 3.2 $h(d, FIN) \leq 2^d$.

Proof: Let $L = \{A_1, \ldots, A_k\} \subset FIN$, $\cap_{i=1}^k C_d(A_i) = \phi$. Consider any $A_i \in L$. It is finite, so $conv(A_i)$ is a polytope, so there exists a finite family of closed halfspaces $\{A_{i,j}\}_j$ such that $conv(A_i) = \cap_j A_{i,j}$. Hence,

$$\phi = \cap_{i=1}^k C_d(A_i) = \cap_{i=1}^k conv(A_i) \cap \mathbf{Z^d} = (\cap_{i,j} A_{i,j}) \cap \mathbf{Z^d},$$

so by Proposition 3.1 there is a subfamily S of $\{A_{i,j}\}_{i,j}$ such that $|S| \leq 2^d$ and $(\cap S) \cap \mathbf{Z^d} = \phi$. Let $I = \{i : \exists j A_{i,j} \in S\}$. Clearly $|I| \leq 2^d$, and $\cap_{i \in I} conv(A_i) = \cap_{i \in I} \cap_j A_{i,j} \subset (\cap S)$, so $\cap_{i \in I} C_d(A_i) = (\cap_{i \in I} conv(A_i)) \cap \mathbf{Z^d} \subset (\cap S) \cap \mathbf{Z^d} = \phi$, so $M = \{A_i : i \in I\} \subset L$ is a subfamily proving the claim.

The following proposition follows immediately from the definitions.

Proposition 3.3 Let (X, C) be a convexity space.

3.3.1 Suppose this space has a Caratheodory number c and an exchange number e, and let $Y \subset X$. Then $(Y, \{A \cap Y : A \in C\})$ is a convexity space (the restriction to Y) with Caratheodory number $c' \leq c$, and exchange number $e' \leq e$.

3.3.2 Let $F_1 \subset F_2 \subset 2^X$. Then

$$c(F_1) \leq c(F_2), \quad h(F_1) \leq h(F_2), \quad r(F_1) \leq r(F_2), \quad e(F_1) \leq e(F_2) \ .$$

3.3.3 Let $F = \{A \subset X : |A| < \infty\}$. Then $r(F) = r(2^X) = r$.

Next we need the following.

Proposition 3.4 ([Sierksma], also [Kay and Womble])
Let (X, C) be a convexity space with finite Caratheodory, Helly and exchange numbers c, h, e, such that $e \leq c$. Then the Radon number satisfies

$$r \leq (c-1)(h-1) + 3.$$

Corollary 3.5 Let (X, C) be a convexity space and let $F \subset 2^X$ be a family of subsets closed under taking subsets.
If there exist two nonnegative integers a, b such that $c(F) \leq a$, $e(F) \leq a$, and $h(F) \leq b$, then

$$r(F) \leq (a-1)(b-1) + 3.$$

Proof: The proof of Proposition 3.4 (see [Sierksma]) remains valid under the assumptions above.

We now give an upper bound on the Radon number $r(d)$ of the convexity space $(\mathbf{Z^d}, C_d)$.

Theorem 3.6 $r(d) \leq d(2^d - 1) + 3$.

Proof: It is known that for $(\mathbf{R^d}, conv)$ the Caratheodory and the exchange numbers are both equal to $d+1$. By Proposition 3.3.1, we then have, for $(\mathbf{Z^d}, C_d)$, that $c(d) \leq d+1$ and $e(d) \leq d+1$, and so by Proposition 3.3.2, $c(d, FIN) \leq d+1$ and $e(d, FIN) \leq d+1$. By Corollary 3.2, $h(d, FIN) \leq 2^d$. Applying now Corollary 3.5 with $a = d+1$, $b = 2^d$ and using Proposition 3.3.3, we get

$$r(d) = r(d, FIN) \leq d(2^d - 1) + 3.$$

Remark The presentation leading to the upper bound of Theorem 3.6 above is based on the result of Bell and Scarf (Proposition 3.1), which is well known in the literature of mathematical programming. In fact, [Doignon] has established that $h(d) = 2^d$, and one could use this result to obtain Theorem 3.6 directly from Corollary 3.5 with $F = 2^X$ (bypassing Corollary 3.2). Furthermore, Proposition 3.1 itself follows easily from this result.

4 Stable Sets in Polytopes

In this section we study in detail some elementary properties of stable sets of vertices of polytopes. We establish the existence of a Radon partition of the set of extreme points (vertices) of a lattice polytope having a large stable set (in the sense defined below) of vertices. As a byproduct, this result yields a simpler and more direct proof of an upper bound for $r(d, SP)$, where $SP = \{A \subset \mathbf{Z^d} : A = ext(P),\ P$ a simple polytope $\}$, which is asymptotically the same as the one given in Theorem 3.6. It is also helpful in proving that $r(2) = 6$.

We start with some notation. By the "graph" of a polytope we mean the abstract graph having as vertices and edges those of the polytope. For two points $x, y \in \mathbf{R^d}$ we denote by $[x, y]$ (resp. (x, y)) the closed (resp. open) line segment joining them. For a vertex $s \in ext(P)$, we denote its set of neighbors by $N_P(s) = \{v \in ext(P) : [s, v]$ is a 1-face of $P\}$. Given a hyperplane H we denote by $H^<, H^>$ (resp. $H^\leq, H^\geq$) its open (resp. closed) supported halfspaces.

In the proofs below we will use some basic facts on polytopes. For convenience, we give references to theorems in [Brondsted].

The following proposition is elementary.

Proposition 4.1 Let P be a d-polytope and H a hyperplane. Let $P' = P \cap H^\leq$, $s \in ext(P) \cap H^<$ and $v \in ext(P) \cap H^\leq$. Suppose $[s, v]$ is an edge of P'. Then it is also an edge of P.

Lemma 4.2 Let P be a d-polytope and $s \in ext(P)$ a vertex having d neighbors. Let $P' = conv(ext(P) \setminus \{s\})$, $H = aff(N_P(s))$, $F = H \cap P$ and $F' = H \cap P'$. Assuming P' is also d-dimensional, the following hold.
4.2.1 H separates s from P'.
4.2.2 F' is a facet of P'.
4.2.3 $F' = F$.
4.2.4 $ext(F') = ext(F) = N_P(s)$.

Proof:
4.2.1 $N_P(s)$ is a set of d affinely independent points, so H is a hyperplane. Now $s \in ext(P) \setminus H$ so H separates s from all its neighbors in P, and so (Thm. 11.8 [Brondsted]) from $ext(P) \setminus \{s\}$.
4.2.2 It follows that H is a supporting hyperplane for P', and $dim(F') = d - 1$, so F' is a facet of P'.
4.2.3 It follows that $ext(F') = ext(P') \cap H = ext(P) \cap H \subset ext(F)$. Suppose indirectly that there exists an $x \in ext(F) \setminus ext(P)$. Then (Thm. 11.1 [Brondsted]) there is a 1-face $[u, v]$ of P such that $\{x\} = [u, v] \cap H$. Since $x \notin \{u, v\} \subset ext(P)$, we must have that one of u, v, say u, equals s. But then $v \in N_P(s) \subset H$, so $\{v\} = [u, v] \cap H = \{x\}$, a contradiction. Thus, $ext(F) \subset ext(P) \cap H = ext(F')$, so $ext(F) = ext(F')$.
4.2.4 We know that $N_P(s) \subset ext(P) \cap H = ext(F)$. For the converse, let $v \in ext(P) \cap H$. Then v is a 0-face of the pyramid $Q = conv(F \cup \{s\})$, so (Thm.7.7 [Brondsted]) $[s, v]$ ia a 1-face of Q. But now we can apply Proposition 4.1: $s \in H^<$, say, and $Q = P \cap H^\leq$, so we conclude that $[s, v]$ is a 1-face of P as well. Thus $v \in N_P(s)$. This completes the proof.

Lemma 4.3 *Stable set exchange lemma*
Let P be a d-polytope, $V = ext(P)$ and $S \subset V$ a stable subset of vertices (in the graph theoretical sense) such that every vertex in S has exactly d neighbors in P. Let $x \in conv(S) \setminus (S \cup conv(V \setminus S))$. Then there exists a vertex $s \in S$ such that in the polytope $P' = conv(V \cup \{x\} \setminus \{s\})$ the following hold.
4.3.1 $ext(P') = V \cup \{x\} \setminus \{s\}$
4.3.2 The graph of P is a subgraph of the graph of P' under the bijection

$$f : ext(P) \to ext(P') : v \to \begin{cases} x & (v = s) \\ v & (v \neq s) \end{cases},$$

and the only possible new edges are between pairs of vertices in $f(N_P(s))$. In particular, the set $S' = S \cup \{x\} \setminus \{s\}$ is stable in P', and each vertex in this set has exactly d neighbors in P'.

Proof:

4.3.1 Let $U \subset V$ be a subset with minimum $|U \cap S|$ such that $x \in conv(U)$. Choose any $s \in S \cap U$. Let $H = aff(N_P(s))$, and $s \in H^<$. Let $P'' = conv(ext(P) \setminus \{s\})$ and $F'' = P'' \cap H$. Assume indirectly that $x \notin H^<$. Then $x \in P''$, so $x \in conv(U \cup ext(F'') \setminus \{s\})$. But $ext(F'') = N_P(s)$ (Lemma 4.2.4) and S is stable so $N_P(s) \cap S = \phi$, so

$$|(U \cup ext(F'') \setminus \{s\}) \cap S| < |U \cap S|,$$

contradicting the choice of U. Hence $x \in H^<$. Now, by Lemma 4.2.1 we know that H separates s from P'', so also separates x from P''. Now $P' = conv(P'' \cup \{x\})$, so $x \in ext(P')$. It is clear that $V \setminus \{s\} \subset ext(P')$.
4.3.2 Let $F = P \cap H$ and $F' = P' \cap H$. Now, $P'' \subset P' \subset P$ and by Lemma 4.2.3 $F = F''$, so $F = F'' \subset F' \subset F$. Thus, $ext(F') = ext(F) = N_P(s)$. Now, take any $v \in N_P(s)$. Then $[x, v]$ is an edge in the pyramid $conv(F' \cup \{x\}) = P' \cap H^{\leq}$, so by Proposition 4.1 it is also an edge of P'. Thus, $N_P(s) \subset N_{P'}(x)$. If, indirectly, there exists a vertex $v \in N_{P'}(x) \setminus N_P(s) = ext(F')$, then $[x, v] \cap H^{\leq}$ is an edge of the pyramid $conv(F' \cup \{x\})$ and so (Thm. 7.7 [Brondsted]) $([x, v] \cap H^{\leq}) \cap H$ is a vertex of F', say u. But then $u \in ext(P')$, so $v = u \in N_P(s)$, a contradiction. Thus $N_{P'}(x) = N_P(s)$.
We now restrict attention to pairs of vertices in $V \setminus \{s\}$. If $[u, v]$ is an edge in P then it is also an edge in $P' \subset P$. If $[u, v]$ is an edge in P' and, say, $u \notin N_P(s)$, then $u \in H^>$ and so $[u, v]$ is an edge of $P'' \subset P'$ as well. Now, $P'' = P \cap H^{\geq}$ so by Proposition 4.1 $[u, v]$ is also an edge of P. This completes the proof.

Theorem 4.4 Let P be a d-lattice polytope such that $S \subset V = ext(P)$ is a stable set of vertices of degree d and $|S| = 2^d + 1$. Then $(S, V \setminus S)$ is a Radon partition of V, i.e. $C_d(S) \cap C_d(V \setminus S) \neq \phi$.

Proof: S is a set of integer points in $\mathbf{Z^d}$ of size $2^d + 1$, so there are two points $y, z \in S$ having the same parity on all coordinates. The point $x = (y + z)/2$ is an integer point in $conv(S) \setminus S$. If $x \in conv(V \setminus S)$ then we are done. If not, then let the point $s \in S$, the polytope P' and the set S' be as defined in Lemma 4.3. Then, by Lemma 4.3, P' and S' satisfy the hypothesis of Theorem 4.4 and we can repeat the argument above. Now $|conv(S') \cap \mathbf{Z^d}| < |conv(S) \cap \mathbf{Z^d}|$ because $s \in conv(S) \setminus conv(S')$, and $|conv(S) \cap \mathbf{Z^d}| < \infty$, so after finitely many applications of the argument above, we obtain a polytope P'' and a set S'' such that $(S'', V \setminus S)$ is a partition of $ext(P'')$ such that there exists an integer point $x'' \in conv(S'') \cap conv(V \setminus S)$. But, by the construction, $conv(S'') \subset conv(S)$, so in fact we have $x'' \in C_d(S) \cap C_d(V \setminus S)$.

Example 4.5 Consider, for $k \geq 3$, the graph $G(k) = (V(k), E(k))$ which is the union of two homeomorphs of the k-wheel (see figure 2), defined as follows (we call it the "Bicycle wheel with $2k$ alternating spokes"):

$$\begin{aligned} V(k) &= \{v, v_0, \ldots, v_{k-1}, u, u_0, \ldots, u_{k-1}\}, \\ E(k) &= \{\{v_i, u_i\} : 0 \leq i \leq k-1\} \cup \{\{u_i, v_{i+1 \ (mod\ k)}\} : 0 \leq i \leq k-1\} \cup \\ &\quad \{\{v, v_i\} : 0 \leq i \leq k-1\} \cup \{\{u, u_i\} : 0 \leq i \leq k-1\}. \end{aligned}$$

The graph $G(k)$ is planar and 3-connected, so by Steinitz's theorem (see, e.g., [Grunbaum], Sec. 13.2) there exists a (convex) 3-lattice polytope $P(k)$ having $G(k)$ as its graph. Note, for example, that the 3-cube C^3 has $G(3)$ as its graph. Now, for $k \geq 9$ and $d \geq 3$, each one of the polytopes $P(k) \times C^{d-3} \subset \mathbf{R^d}$ satisfies the hypothesis of Theorem 4.4, and so admits a Radon partition. Note that for $k = 9$ or 10 we can not deduce this from Theorem 3.6.
As a corollary to Theorem 4.4 we get again the upper bound, for simple polytopes, and so we have:

Corollary 4.6 $2^d + 2^{d-2} + 1 \leq r(d, SP) \leq d2^d + 2$.

Proof: The lower bound is the same as the one given in Corollary 2.2, as the proofs in Section 2 remains valid for simple polytopes. For the upper bound, recall that the graph of a simple polytope is d-regular, so if the polytope has at least $d2^d + 2$ vertices, it has a stable set of size at least $\lceil ((d2^d + 2) - 1)/d \rceil = 2^d + 1$. Then, by Theorem 4.4, the set $ext(P)$ has a Radon partition.

5 The Two Dimensional Case

In Section 2 we have shown that $r(2) \geq 6$. We shall now prove the converse. We will use the following fact (see, e.g., [Gruber and Lekkerkerker], Thm.4, p.20).

Proposition 5.1 *Theorem on Lattice Triangles*
Let v_1, v_2 be two linearly independent points in $\mathbf{Z^2}$. If

$$C_2(\{(0,0), v_1, v_2\}) = conv(\{(0,0), v_1, v_2\}) \cap \mathbf{Z^2} = \{(0,0), v_1, v_2\},$$

Then the set $\{v_1, v_2\}$ is a lattice basis of $\mathbf{Z^2}$.

Lemma 5.2 Let $V \subset \mathbf{Z^2}$ be such that $|V| = 6$ and $P = conv(V)$ is a convex hexagon. Let $S \subset V$ be a stable subset with $|S| = 3$. Then $C_2(S) \setminus S \neq \phi$ (recall $C_2(S) = conv(S) \cap \mathbf{Z^2}$).

Proof: Without loss of generality we may assume that $S = \{(0,0), v_1, v_2\}$. Consider the two lines $l_i = lin(v_i)$ $(i = 1, 2)$. The four connected components of $\mathbf{R^2} \setminus (l_1 \cup l_2)$ are (see figure 3)

$$c_{i,j} = cone(\{(-1)^i v_1, (-1)^j v_2\}) \setminus (l_1 \cup l_2) \quad (i, j = 0, 1).$$

Let $W = V \setminus S$. No three points of V are colinear, and $|l_i \cap S| = 2$ $(i = 1, 2)$, so $l_i \cap W = \phi$ $(i = 1, 2)$. Also, P is a convex hexagon and S is stable, so W is stable as well, and so no two points of W lie on the same component $c_{i,j}$. Thus, there exist two points $w_1, w_2 \in W$ lying on adjacent components, say $w_1 \in c_{1,0}$, $w_2 \in c_{0,0}$.
Now, suppose indirectly that $C_2(S) = S$. Then, by Proposition 5.1, $\{v_1, v_2\}$ is a basis of the lattice $\mathbf{Z^2}$. Now for $i = 1, 2$, $w_i \in \mathbf{Z^2}$ so lies on the line $l_1 + k_i v_2$ for some integer $k_i \geq 1$. But then we have $v_2 \in conv(\{(0,0), w_1, w_2\})$, a contradiction.

Theorem 5.2 $r(2) = 6$.

Proof: Let $U \subset \mathbf{Z^2}$ be such that $|U| \geq 6$. We may assume that there is a subset $V \subset U$ such that $|V| = 6$ and $P = conv(V)$ is a convex hexagon. It is enough to show that V admits a Radon partition. Let $S \subset V$ be a stable set with $|S| = 3$. By Lemma 5.2, $C_2(S) \setminus S \neq \phi$. Then either $C_2(S) \cap C_2(V \setminus S) \neq \phi$, in which case we are done, or there exists a point $x \in C_2(S) \setminus (S \cup conv(V \setminus S))$. In that case, we can do a stable set exchange and obtain a new set S' and polytope P' as defined in Lemma 4.3, and in particular $conv(S') \subset conv(S)$. Repeating the same argument as above, we eventually get, as in the proof of Theorem 4.4, a set S'' such that $|S''| = 3$, $P'' = conv((V \setminus S) \cup S'')$ is a convex hexagon, S'' is stable in P'' and $C_2(S'') \cap C_2(V \setminus S) \neq \phi$. But then we are done, since $conv(S'') \subset conv(S)$ and so $(S, V \setminus S)$ is a Radon partition of V.

6 Computational Complexity Aspects

Consider the following decision problems.

RADON(d):
Instance: $n \in N$ and $A \subset \mathbf{Z^d}$ such that $|A| = n$.
Question: Does A admit an (integral) Radon partition?

RADON:
Instance: $d, n \in N$ and $A \subset \mathbf{Z^d}$ such that $|A| = n$.
Question: Does A admit an (integral) Radon partition?

Note that in RADON(d), the dimension d is fixed and is not part of the input. We show that RADON(d) is decidable in polynomial time, and in fact, as a result of Theorem 3.6, in constant number of arithmetic operations, while, in contrast, RADON is NP-complete.

Proposition 6.1 RADON(d) is decidable in polynomial time.

Proof: Given an input to RADON(d), if $n \geq r(d)$ then there exists a Radon partition. Otherwise, consider all possible pairs (S,T) of disjoint nonempty subsets of the input set A such that $|S|, |T| \leq d+1$ and the points in S (resp. T) are affinely independent in $aff(S)$ (resp. $aff(T)$). Since the Caratheodory number of the integers $c(d) \leq d+1$, there will be such a pair with $C_d(S) \cap C_d(T) \neq \Phi$ if and only if the input set A has a Radon partition. The number of such pairs is bounded by a polynomial function of n, and in fact, by Theorem 3.6, $n < r(d) < \infty$, so it is bounded by a finite constant which depends on d only. For each pair (S,T) with $|S| \leq d+1, |T| \leq d+1$, do the following. Find a system of linear equalities defining $aff(S)$, and check affine independence, i.e., $|S| = 1 + dim(aff(S)$ (otherwise, move on to the next pair). Next, add inequalities to this system to get a description of $conv(S)$. This is easy, as $conv(S)$ is a simplex in $aff(S)$, so any $(|S|-1)$-subset of S spans a facet of $conv(S)$ in $aff(S)$.

It is easily verified that the bit size of such a description is bounded above by a polynomial function in the bit size of the input, and so this step could be done in polynomial time.

Similarly, find a linear inequalities and equalities description of $conv(T)$.

Now let L be the linear program which is the union of the two systems of inequalities found above. Apply to it any algorithm that checks if its solution set contains an integer point, e.g. the integer programming algorithm given in [Lenstra], which runs in polynomial time when the dimension d is not a part of the input. Since the solution set of L is $conv(S) \cap conv(T)$, this algorithm checks if $C_d(S) \cap C_d(T) \neq \Phi$, which is what we need.

Remark: It follows that RADON(d) is decidable in constant number of arithmetic operations.

Next, we show that, in contrast, RADON is NP-complete. The following problem is known to be NP-complete (see [Garey and Johnson],p.223).

PARTITION:
Instance: $n \in N$ and a set of nonnegative integers $B = \{a_1, \ldots, a_{2n}\}$.
Question: Is there a balanced partition, namely a partition (I,J) of $[2n] = \{1, \ldots 2n\}$ such that $|I| = |J| = n$ and $\sum_{i \in I} a_i = \sum_{j \in J} a_j$?

Theorem 6.2 RADON is NP-complete.

Proof: To show that RADON is in NP, suppose the instance in question has a Radon partition (S,T). Then there exists $x \in C_d(S) \cap C_d(T)$, and so the triple (x,S,T) is a witness of size bounded by a polynomial in the size of the input, to the existence of a Radon partition, which can be checked in polynomial time.

We now show that PARTITION$\leq_m^P$RADON. Given an input (n,B) to PARTITION, let $d = 2\binom{2n}{2}+1, k = n+n^2$. We construct a set A of $2n+2\binom{2n}{2}$ points in $\mathbf{Z^d}$ such that A has a Radon partition if and only there exists a balanced partition for (n,B). For convenience, we index the first coordinate of a point by 0, and for each pair i,j such that $1 \leq i < j \leq 2n$, a point will have two coordinates, one indexed by $1,i,j$ and the other by $2,i,j$, so a point will be written as

$$x = (x_0, x_{1,1,2}, x_{2,1,2}, x_{1,1,3}, x_{2,1,3}, \ldots, x_{1,2n-1,2n}, x_{2,2n-1,2n}).$$

Now, for each $i \in [2n]$, we will have one point x^i, and for each pair i,j such that $1 \leq i < j \leq 2n$ we will have two points $y^{i,j}, z^{i,j}$, so

$$A = \{x^1, \ldots, x^{2n}, y^{1,2}, z^{1,2}, \ldots, y^{2n-1,2n}, z^{2n-1,2n}\},$$

where the points are defined as follows.
$\forall i \in [2n]$ let $x_0^i = ka_i$.
$\forall i,j \in [2n]$ such that $i < j$, let

$$x_{1,i,j}^i = y_{1,i,j}^{i,j} = -1, \quad x_{1,i,j}^j = z_{1,i,j}^{i,j} = 1, \quad y_{2,i,j}^{i,j} = z_{2,i,j}^{i,j} = k.$$

All other entries of all points will be set to zero.

Clearly the set A defined that way could be constructed from B in polynomial time.

Now, suppose first that (I, J) is a balanced partition for (n, B). Define $S, T \subset A$ as follows.

$$S = \{x_i : i \in I\} \cup \{z_{i,j} : 1 \leq i < j \leq 2n, i \in I, j \in J\} \cup \{y_{i,j} : 1 \leq i < j \leq 2n, i \in J, j \in I\},$$

$$T = \{x_i : i \in J\} \cup \{y_{i,j} : 1 \leq i < j \leq 2n, i \in I, j \in J\} \cup \{z_{i,j} : 1 \leq i < j \leq 2n, i \in J, j \in I\}.$$

It is easy to see that $S \cap T = \Phi$ and $|S| = |T| = k$, and that $\sum_{s \in S} \frac{1}{k} s = \sum_{t \in T} \frac{1}{k} t \in \mathbf{Z^d}$, so $C_d(S) \cap C_d(T) \neq \Phi$, and so, for instance, $(S, A \setminus S)$ is a Radon partition.

Conversely, suppose (S, T) is a Radon partition of A, i.e. $\sum_{s \in S} c(s)s = \sum_{t \in T} c(t)t \in \mathbf{Z^d}$, where $\forall x \in A\ 0 \leq c(x) \leq 1$ and $\sum_{s \in S} c(s) = \sum_{t \in T} c(t) = 1$, and for some $s \in S, t \in T$ we have $c(s) \neq 0, c(t) \neq 0$.

Let $1 \leq i < j \leq 2n$, and consider the $(2, i, j)$-th coordinate. It is clear, then, that if $c(y^{i,j}) > 0$ or $c(z^{i,j}) > 0$, then in fact $c(y^{i,j}) = c(z^{i,j})$ and $y^{i,j}, z^{i,j}$ are on opposite sides of the partition. If both coefficients above are zero, then, moving $y^{i,j}$ to the other side if necessary, we may assume again that it is on the opposite side of $z^{i,j}$. So, we may assume that for all $1 \leq i < j \leq 2n$ we have $c(y^{i,j}) = c(z^{i,j})$ and $y^{i,j}, z^{i,j}$ are on opposite sides of the partition.

Now let $1 \leq i < j \leq 2n$, and consider the $(1, i, j)$-th coordinate. There are four cases to be considered. (1) $x^i, x^j, y^{i,j}$ are on the same side of the partition and $z^{i,j}$ is on the other side. Then we have

$$\begin{aligned} -c(x^i) + c(x^j) - c(y^{i,j}) = c(z^{i,j}) \in \mathbf{Z} &\implies \\ 0 \leq 2c(z^{i,j}) = c(z^{i,j}) + c(y^{i,j}) = c(x^j) - c(x^i) \leq 1 &\implies \\ 1 > c(z^{i,j}) \in \mathbf{Z} \implies c(y^{i,j}) = c(z^{i,j}) = 0 &\implies \\ c(x^i) = c(x^j). & \end{aligned}$$

(2) $x^i, x^j, z^{i,j}$ are on the same side of the partition. and $y^{i,j}$ is on the other side. Then we have

$$\begin{aligned} -c(x^i) + c(x^j) + c(z^{i,j}) = -c(y^{i,j}) \in \mathbf{Z} &\implies \\ 0 \leq 2c(y^{i,j}) = c(z^{i,j}) + c(y^{i,j}) = c(x^i) - c(x^j) \leq 1 &\implies \\ 1 > c(y^{i,j}) \in \mathbf{Z} \implies c(z^{i,j}) = c(y^{i,j}) = 0 &\implies \\ c(x^i) = c(x^j). & \end{aligned}$$

(3) $x^i, y^{i,j}$ are on the same side of the partition and $x^j, z^{i,j}$ are on the other side. Then we have

$$0 \geq -c(x^i) - c(y^{i,j}) = c(x^j) + c(z^{i,j}) \geq 0 \implies c(x^i) = c(x^j) = c(z^{i,j}) = c(y^{i,j}) = 0.$$

(4) $x^i, z^{i,j}$ are on the same side of the partition and $x^j, y^{i,j}$ are on the other side. Then we have

$$-c(x^i) + c(z^{i,j}) = c(x^j) - c(y^{i,j}) \in \mathbf{Z}.$$

If $c(y^{i,j}) = 0$, then

$$c(z^{i,j}) = 0 \implies 0 \geq -c(x^i) = c(x^j) \geq 0 \implies c(x^i) = c(x^j) = 0.$$

If $c(y^{i,j}) = 1$, then

$$c(z^{i,j}) = 1 \implies 0 \leq 1 - c(x^i) = c(x^j) - 1 \leq 0 \implies c(x^i) = c(x^j) = 1.$$

Otherwise, $|c(x^j) - c(y^{i,j})| < 1$ and $c(x^j) - c(y^{i,j}) \in \mathbf{Z}$, so

$$c(x^j) - c(y^{i,j}) = 0 \implies -c(x^i) + c(z^{i,j}) = 0 \implies c(x^i) = c(z^{i,j}) = c(y^{i,j}) = c(x^j).$$

Summing up the cases, we have in (1) and (2) that $c(x^i) = c(x^j)$ and $c(y^{i,j}) = c(z^{i,j}) = 0$, while in (3) and (4) we have $c(x^i) = c(x^j) = c(y^{i,j}) = c(z^{i,j})$. Hence, if for all $i \in [2n]$ we have $c(x^i) = 0$, then also

for all i, j such that $1 \leq i < j \leq 2n$ we have $c(y^{i,j}) = c(z^{i,j}) = 0$, which is impossible. Hence there exists $i \in [2n]$ such that $c = c(x^i) > 0$. But for all $1 \leq i < j \leq 2n$ we have $c(x^i) = c(x^j)$, so for all $i \in [2n]$ we get $c(x^i) = c > 0$ (so, in fact, case (3) can not occur).

Now, let $I = \{i : x^i \in S\}$, $J = \{j : x^j \in T\}$. Consider $1 \leq i < j \leq 2n$. If $|\{i,j\} \cap I| = 0$, then case (1) or (2) holds, so $c(y^{i,j}) = c(z^{i,j}) = 0$. If $|\{i,j\} \cap I| = 1$, then (4) holds, so $c(y^{i,j}) = c(z^{i,j}) = c(x^i) = c$. Let

$$l = |\{\{i,j\} : |\{i,j\} \cap I| = 1,\ 1 \leq i < j \leq 2n\}|.$$

Since $y^{i,j}, z^{i,j}$ are on opposite sides of the partition, we get

$$c(|I| + l) = \sum_{s \in S} c(s)s = \sum_{t \in T} c(t)t = c(|J| + l),$$

so $|I| = |J|$, so (I, J) is a partition of $[2n]$ such that $|I| = |J| = n$.

Finally, consider the 0-th coordinate. We have

$$\sum_{i \in I} cka_i = \sum_{i \in I} cx_0^i = \sum_{s \in S} c(s)s_0 = \sum_{t \in T} c(t)t_0 = \sum_{j \in J} cx_0^j = \sum_{j \in J} cka_j \Longrightarrow \sum_{i \in I} a_i = \sum_{j \in J} a_j,$$

so (I, J) is a balanced partition for (n, B).

7 Remaining Questions

Computing the lower and upper bounds given in Corollary 2.2 and Theorem 3.6, we obtain $11 \leq r(3) \leq 24$. For simple polytopes we can do a little better.

Proposition 7.1 $11 \leq r(3, SP) \leq 21$.

Proof: Let P be a 3-simple polytope, and let f_i be the number of i-faces of P $(i = 0, 1, 2)$. Suppose $f_0 \geq 21$. Let $g(i)$ be the number of vertices of P contained in its i-th 2-face $(i = 1, \ldots, f_2)$. We claim that there exists a 2-face having $g(i) \geq r(2) = 6$. Since P is simple, we have $f_1 = 3f_0/2$, and so Euler's formula becomes

$$f_2 = 2 + f_1 - f_0 = 2 + f_0/2 \geq 2 + 21/2,$$

so $f_2 \geq 13$. Now, each vertex lies in exactly three 2-faces, so counting incidences of 0-faces and 2-faces, we get

$$\sum_{i=1}^{f_2} g(i) = 3f_0 = 3(2(f_2 - 2)) = 6f_2 - 12,$$

and so

$$max\{g(i) : 1 \leq i \leq f_2\} \geq (\sum_{i=1}^{f_2} g(i))/f_2 = (6f_2 - 12)/f_2 \geq 6 - 12/13 > 5,$$

which proves the claim. But then the set of vertices of this face has a Radon partition, by Observation 1.5, so the set $ext(P)$ has a Radon partition as well.

It will be interesting to find tighter lower and upper bounds for $r(d)$ and $r(d, SP)$, or at least the exact values of $r(3)$ and $r(3, SP)$. Even more specific, does the set of extreme points of any 3-lattice polytope having the Dodecahedron as its graph (see figure 4), always admit a Radon partition?

8 Acknowledgement

I am grateful to Leslie E. Trotter, Jr., for many discussions which led me both into asking the question about the Radon number for the integer lattice and into providing a partial solution to it, and for his careful reading of the manuscript. I also thank Gunter M. Ziegler and Joseph S. B. Mitchell for helpful remarks concerning the presentation of this paper.

References

[1] A. Brondsted, **An Introduction to Convex Polytopes**, Springer-Verlag, 1983.

[2] W. Cook, J. Fonlupt and A. Shrijver, *An Integer Analogue of Caratheodory's Theorem*, Journal of Combinatorial Theory (B), Vol. 40, 1986, p. 63-70.

[3] L. Danzer, B. Grunbaum and V. Klee, *Helly's theorem and its relatives*, American Mathematical Society, Proceedings of Symposia in Pure Mathematics, Vol. 7, 1963, p. 101-180.

[4] J. P. Doignon, *Convexity in Crystallographical Lattices*, Journal of Geometry, Vol. 3, 1973, p. 71-85.

[5] J. Eckhoff, *Der Satz von Radon in Konvexen Produktstrukturen I*, Monatshefte fur Mathematik, Vol. 72, 1968, p. 303-314.

[6] M. R. Garey and D. S. Johnson, **Computers and Intactability**, Freeman, 1979.

[7] P. M. Gruber and C. G. Lekkerkerker, **Geometry of Numbers**, North Holland, 1987.

[8] B. Grunbaum, **Convex Polytopes**, Wiley, 1967.

[9] P. C. Hammer, *Extended Topology: Caratheodory's Theorem on Convex Hulls*, Rendiconti del Circulo Mathematico de Palermo, Vol. 14, 1965, p. 34-42.

[10] D. C. Kay and E. W. Womble, *Axiomatic Convexity Theory and Relationships Between the Caratheodory, Helly and Radon Numbers*, Pacific Journal of Mathematics, Vol. 38, No. 2, 1971, p. 471-485.

[11] H. W. Lenstra Jr., *Integer Programming with a Fixed Number of Variables*, Mathematics of Operation Research, Vol. 8, 1983, p. 538- 548.

[12] F. W. Levi, *On Helly's Theorem and the Axioms of Convexity* , Journal of Indian Mathematical Society, Vol. 15, 1951, p. 65-76.

[13] A. Shrijver, **Theory of Linear and Integer Programming**, Wiley, 1986.

[14] G. Sierksma, *Relationships Between Caratheodory, Helly, Radon and exchange Numbers of Convexity Spaces*, Nieuw Archief voor Wiskunde, Vol. 3, No. 25, 1977, p. 115-132.

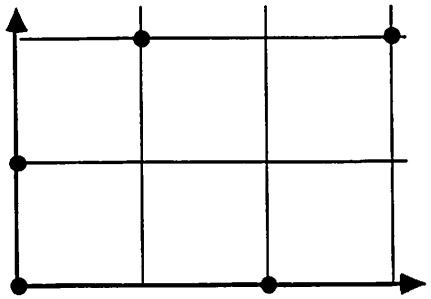


Figure 1

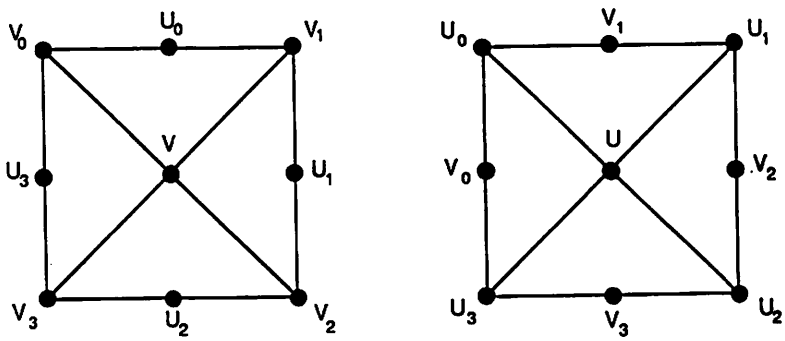


Figure 2

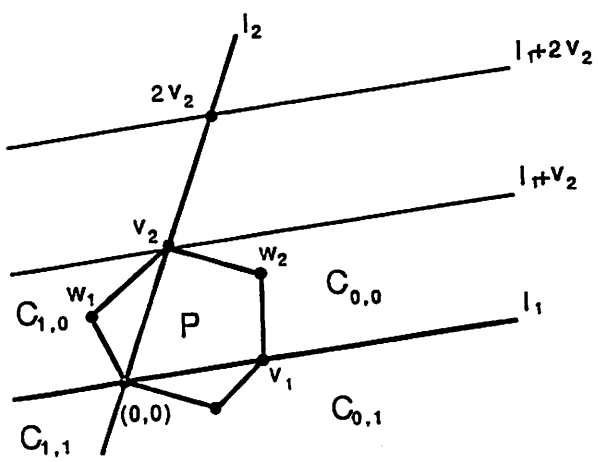


Figure 3

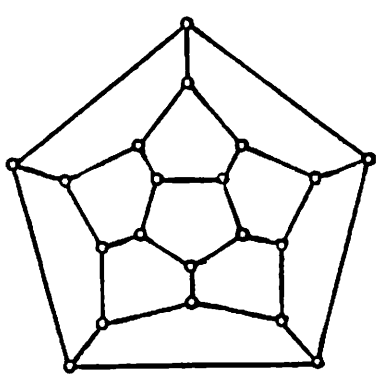


Figure 4

VERTEX DISJOINT CHANNEL ROUTING ON TWO LAYERS

András Recski
Dept. Comp. Sci., L. Eötvös Univ., H-1088 Budapest, Hungary

and Frank Strzyzewski
Fac. El. Eng.,Techn. Univ. Budapest, H-1521 Budapest, Hungary

Abstract: Channel routing is the simplest nontrivial subtask of the problem of routing printed circuit boards or integrated circuits. Many papers were published concerning the channel routing on two layers in the "Manhattan" model (where horizontal and vertical wire segments are positioned on different sides of the board); the problem is known to be NP-complete. We show that in the unconstrained model (where both sides of the board can be used for any direction) every specification is realizable (this was first found by Marek-Sadowska and Kuh, 1983) and give a linear time algorithm for this. The problem of determining the minimum width for a given specification is open; we present some partial results, including a minimax result for the minimum width of the "double river routing" problem if the specification is dense and consists of 2-terminal nets only.

A channel is a rectangular grid G of tracks (numbered from 0 to w+1) and columns (numbered from 1 to m), where w is the width and m the length of the channel. A net $N=(\{t_1,\dots,t_u\},\{b_1,\dots,b_\ell\})$ is a collection of terminals, where the lower terminal b_j is located at the grid point $(b_j,0)$ on track 0 (at the lower boundary) and the upper terminal t_j at the grid point $(t_j,w+1)$ on track w+1 (at the upper boundary). N is called 2-terminal if $u=\ell=1$, and multiterminal otherwise. A 2-terminal net ($\{t\}$, $\{b\}$) or shortly (t, b) is of shape / if $t \geq b$ and of shape \ if $t < b$.

A channel routing problem (CRP) is a set of pairwise disjoint nets $\mathcal{N}=\{N_1,\dots,N_n\}$. A CRP is dense if every boundary point belongs to some nets. The solution (also called layout) of a CRP is a set $\mathcal{G}=\{G_1,\dots,G_n\}$ of subgraphs G_i of G (also called wires) such that G_i connects the terminals of N_i, for $i=1,2,\dots,n$ under the conditions of the corresponding wiring model.

The wiring models are formulated as restrictions on the mutual relations of the subgraphs. The simplest approach is the single layer model (SLM) where the wires are

vertex-disjoint. As a special case of the 2-layer problem, many people studied the so called Manhattan-model (MM) where the wires are edge disjoint and crossings of shape + are permitted but using a common vertex with a shape ⌟ (also called knock-knee) is prohibited. (Such layouts can clearly be realized on two layers, one for the horizontal and one for the vertical edges.) An obvious generalization of MM is the edge disjoint model (EDM) where knock-knees are also permitted.

Finally, the unrestricted 2-layer model (ULM) requires vertex-disjoint wires again, however, the initial graph G is not a planar rectangular grid but an mX(w+2)X2 cubic grid. In this case a lower or upper terminal is a vertex-pair $\{(b_j,0,0), (b_j,0,1)\}$ or $\{(t_j,w+1,0), (t_j,w+1,1)\}$, respectively, together with the edge connecting the two vertices. Obviously, SLM is a special case of MM and this latter is a special case of EDM. It is also clear that every solution of MM corresponds to a solution of ULM. EDM and ULM are incomparable, however, see p. 153 of (Johnson, 1984).

The two basic problems of channel routing are as follows:

(P1) Decide whether a channel routing problem is soluble.

(P2) If yes, determine a layout with minimum width.

Some other objectives can also be of importance (like minimizing total wiring length or total number of via holes) but are disregarded in the present paper.

In the SLM the answer to (P1) is yes if and only if no two nets are crossing, that is, if $i<j$ implies that every upper terminal of net N_i preceds every upper terminal of N_j and every lower terminal of N_i preceds every lower terminal of N_j. If this condition holds and every net is 2-terminal (also called river routing, see Dolev et al., 1981) then the minimum width of the layout (the answer to (P2)) is the smallest number k satisfying the property

$$k \geq n \text{ or } t_p \geq b_{p-k}+k \text{ for } p=k+1,\ldots,n \qquad (1)$$

The answer to (P1) in the EDM is given by Frank (1982) and is NP-complete in the MM (LaPaugh, 1980 and Symanski, 1982).

In the present paper

--we characterize those systems of 2-terminal nets which posses layouts without via holes ("double river routing") and prove a minimax result for their minimum width in the

dense case (Theorem 1.2),

--we characterize those dense systems of 2-terminal nets whose layouts have minimum width 1 or 2 (Theorem 2.1) and give some conditions for those with minimum width 3 (Theorems 2.2-2.3),

--we prove that the answer to (P1) is always yes (this result was first obtained by Marek-Sadowska and Kuh, 1983), and

--we give a linear time algorithm for such a layout (which, generally, is not of minimum width), see Theorem 3.1.

I. A minimax theorem

Remark: Throughout (except in Figures 10-12), ULM layouts are illustrated by two drawings, one for the top layer and one for the bottom layer. Terminals and via holes are denoted by small squares.

Lemma 1.1 If a dense 2-terminal CRP is soluble with width 1 or 2 then no via hole is required.

Proof. Suppose that a layout contains a via hole V next to a terminal node N. By the density N must be connected to V on, say, the first side of the board and then a line leaves V to some direction on the second side. Delete V and replace the line section NV from the first side to the second one. □

Remark: Density is clearly a necessary condition here; so is the condition that every net be 2-terminal: Figures 1 and 2 show the layout of two CRP s which clearly cannot be solved without vias since there are three nets so that any two of them must intersect. However, density alone is enough if the CRP is soluble with width 1.

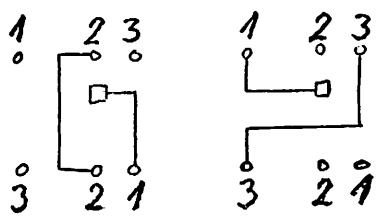


Figure 1

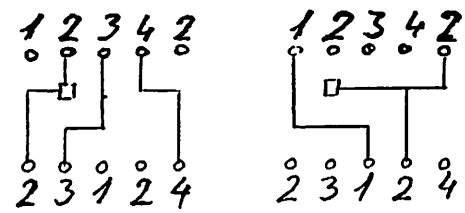


Figure 2

This observation motivates our study in the via-free layouts of dense 2-terminal CRP's.

We call two 2-terminal nets (t_i,b_i) and (t_j,b_j) intersecting if $(t_i-t_j)(b_i-b_j)<0$ and define the constraint graph of a 2-terminal CRP so that the vertices are the nets and intersecting nets become adjacent. For example, the constraint graph of the 2-terminal CRP $\{(1,4),(2,2),(3,6),(4,1),(5,5),(6,3)\}$ is shown in Figure 3.

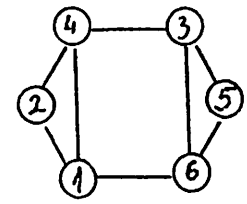


Figure 3

Remark: These graphs are just the permutation graphs, see (Golumbic, 1980), for example. Due to our definition of intersecting, they need not be interval graphs, hence they may contain chordless circuits, like (1,4,3,6) in Figure 3. However, they cannot contain chordless circuits of length ≥ 5, see Lemma 1.3 in (Recski, 1989).

Theorem 1.2 A dense 2-terminal CRP is soluble on 2 layers without via holes if and only if its constraint graph G is bipartite. The minimal width w equals the maximum degree $\Delta(G)$ in G.

Proof. The first statement is trivial: Bipartiteness is necessary for via-free routing on two layers and the two bipartition classes determine two separate river routing problems.

In order to prove $\Delta \geq w$ we suppose that $w>k$ for some k. That means that on one of the two sides of the board, say on the one containing the nets of shape \, $t_p<b_{p-k}+k$ holds for some $p>k$, see Eq. (1). Let α and $\beta=n-\alpha$ denote the number of nets of shape \ and /, respectively. Clearly, $b_{p-k}-1=p-k-1+x$ and $n-t_p=(\alpha-p)+y$, where x is the number of nets of shape / with lower terminal less than b_{p-k} and y is the number of nets of shape / with upper terminal greater than u_p. By the initial assumption, $t_p=n-y-(\alpha-p)<(p-k-1+x)+k+1$, leading to $x+y>n-\alpha$. The right hand side is just β, hence there exists a net of shape / which intersects all the nets p-k, p-k+1,..., p. The corresponding vertex in G has degree at least k+1, hence $\Delta>k$ also holds, as requested.

On the other hand, suppose that $\Delta>w$, i.e., that there exists a net $N=(\{x_2\},\{x_1\})$ of shape, say, / (that is, $x_1<x_2$) which intersects the nets $N_j=(\{t_j\},\{b_j\})$ for $j=i,i+1,\ldots,i+\Delta-1$. Exactly these nets of shape \ satisfy $b_j>x_1$ and $t_j<x_2$ simultaneously, hence $x_2-x_1=\Delta$ (since no two nets of shape / intersect). By Eq. (1) we have $t_p\geq b_{p-w}+w$ for every $p>w$, leading to

$$x_1<b_i\leq t_{i+w}-w\leq t_{i+w+1}-w-1\leq\ldots\leq t_{i+\Delta-1}-(\Delta-1)<x_2-(\Delta-1).$$

Thus $x_2-x_1\geq\Delta+1$, a contradiction. □

II. Dense 2-terminal CRP with small width

As a corollary of Theorem 1.2 we immediately obtain

Theorem 2.1 Let P be a dense 2-terminal CRP and let G be its constraint graph. Then

-- P is soluble on two layers with width 1 if and only if G consists of isolated vertices and vertex-disjoint edges only, and

-- P is soluble with width 2 if and only if G consists of vertex disjoint paths (possible isolated vertices included) and even circuits only. □

These situations are illustrated on Figures 4 and 5, respectively.

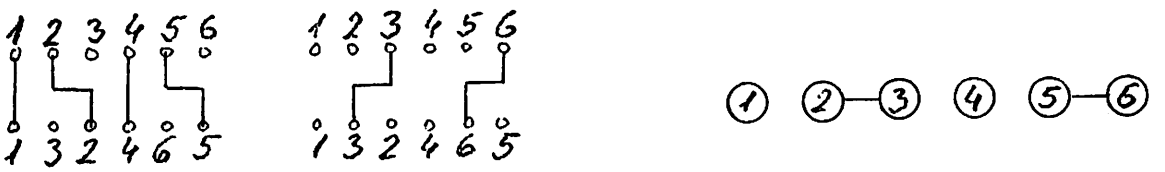


Figure 4

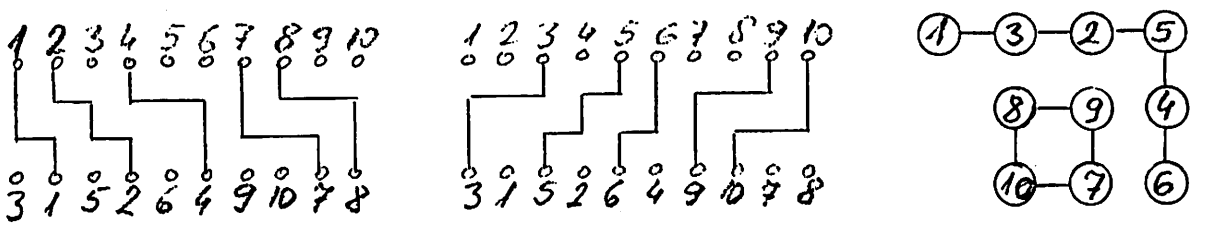


Figure 5

In case of width ≥ 3 the constraint graph need not be bipartite any more. However, if a dense 2-terminal CRP is soluble with width 3, the shaded area of the board (see Figure 6) need not contain via holes (this can be proved in the same way as Lemma 1.1). Moreover, the following result is also true:

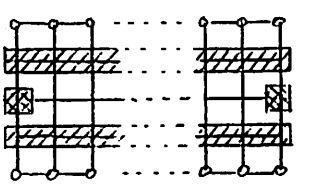


Figure 6

Theorem 2.2 If a dense 2-terminal CRP is soluble with width 3 then, in addition to the shaded area, the two ends of the central track (doubly shaded area on Figure 6) need not contain via holes either.

Its proof is somewhat lengthy, the reader is referred to (Recski, 1989; Theorem 3.1). □

Let L be a vertical line separating some columns from the rest. Its congestion c(L) is the number of nets which are separated by L. For example, there are four essentially different lines in case of the CRP of Figure 2, with respective congestions (from left to right) 2, 3, 1 and 2.

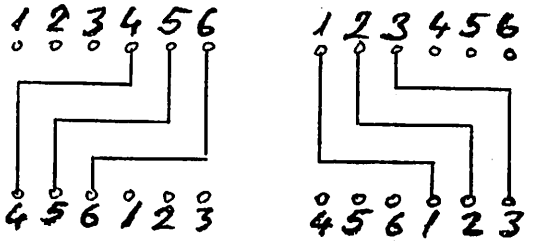


Figure 7

If a CRP is dense and 2-terminal then c(L) is even and if the CRP is soluble with width w then $c(L) \leq 2w$, for every L. This bound is sharp: consider the CRP $\{N_1, N_2, \ldots, N_{2k}\}$ with $N_i = (\{i\}, \{i+k\})$ if $i \leq k$ and $N_i = (\{i-k\}, \{i\})$ if $i > k$; this has a constraint graph $K_{k,k}$ and is soluble with width k, see Figure 7 if k=3.

With a fairly long and technical proof (see Theorems 3.2 through 3.5 in (Recski, 1989)) one can obtain the following results.

Theorem 2.3 Suppose that a dense 2-terminal CRP is soluble with width 3. Then

(1) if L_1, L_2 are two vertical lines, separated by columns, and $c(L_1)=c(L_2)=6$ then their distance is at least 5;

(2) if c(L)=6 then the distance of L from the nearest via hole is greater than 2;

(3) if L separates two adjacent via holes then either c(L)=2 or the same CRP is soluble by a different layout, not requiring these two via holes. □

A combination of these results proves the truth of the following conjecture for w=3. Let n(L) be the number of via holes adjacent to a vertical line L. Clearly, $n(L) \leq 2(w-2)$ can be obtained for a dense 2-terminal CRP, by Lemma 1.1.

Conjecture 2.4 If a dense 2-terminal CRP is soluble with width w then there exists a layout with

$$c(L)+2n(L) \leq 2w \text{ for every } L.$$

III. A linear time algorithm for ULM channel routing

Theorem 3.1. Every channel routing problem is soluble in the above 2-layer model (ULM).

Proof. At first we give a linear time algorithm for the case if every net is 2-terminal. The steps of the algorithm will be illustrated for the CRP $\{N_1,\ldots,N_9\}$ where the nets N_i are (1,3), (2,8), (3,5), (4,2), (5,7), (6,4), (7,9), (8,1) and (9,6), respectively.

Step 1 Arrange the nets of shape \ in decreasing order of their second coordinate.	(7,9), (2,8), (5,7), (3,5), (1,3).
Step 2 Arrange the nets of shape / in increasing order of their first coordinate.	(4,2), (6,4), (8,1), (9,6).
Step 3 Realize the lower vertical segment of every net and the medium segment of the nets of shape \ on the first layer.	see the first five horizontal lines in Figure 8
Step 4 Realize the upper vertical segment of every net and the medium segment of the nets of shape / on the second layer.	see the last four horizontal lines in Figure 8

Step 5 If $t_i=b_i$ for some nets N_i then a free horizontal line is required between Steps 3 and 4 for the via holes of these nets.

In case of multiterminal nets realize the horizontal part of the upper interconnections on the first layer and that of the lower ones on the second layer (see Figure 9 as an example).

Remark 3.2 The running time for the algorithm is linear in n. The complexity of the sortings is O(n) rather than O(nlogn) since the n objects are just the integers between 1 and n.

Remark 3.3 The obtained layout is clearly not optimal. Even if we keep the homotopy of each wire, we can make it more compact in a straightforward way (reducing the width from 9 to 6 in Figure 8 and from 10 to 8 in Figure 9 and, at the same time, eliminating about half of the via holes), see Figures 10 and 11.

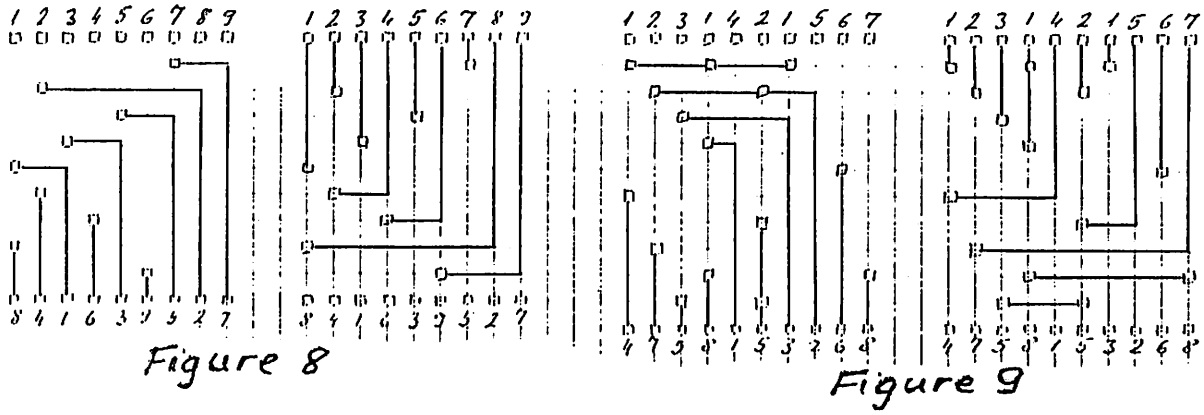


Figure 8

Figure 9

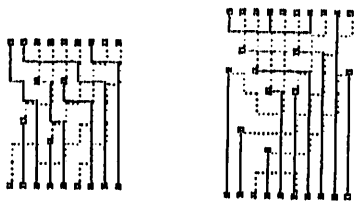


Figure 10

Figure 11

Remark 3.4 Although we are not going into computational details, we present the (compactified) version of a more complex problem in Figure 11. The specification was obtained from previous publications of some authors at Bell Labs (Kernighan et al., 1973), see also (Yoshimura and Kuh, 1982). The theoretical minimum width in the Manhattan model is 18 (and may be less in the unconstrained model), our algorithm yields 22.

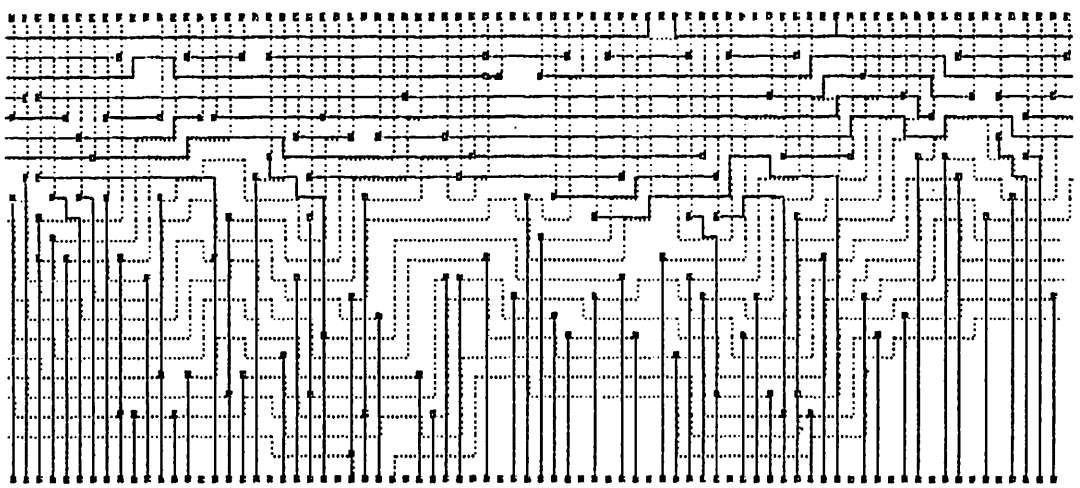


Figure 12

Remark 3.5 A possible disadvantage of this kind of layout is that long sections of parallel line segments arise. This can lead to "cross talk" and is forbidden by some authors (unit-vertical-overlap model, see (Gao and Kaufmann, 1987) for example).

Acknowledgements: Part of the research of the first author was performed when he was visiting the Institute for Operations Research, Bonn, F. R. G. The work was partially supported by the Hungarian Academy of Sciences (Contract No. OTKA 1059) and by the Alexander-von-Humboldt Foundation. Useful conversations with I. Abos, G. Bacsó, V. Chvátal, A. Frank, B. Korte, M. Middendorf and A. Sebő are gratefully acknowledged.

References

A. Frank, 1982. Disjoint paths in a rectilinear grid, Combinatorica 2, 361-371.

D. Dolev, K. Karplus, A. Siegel, A. Strong and J. D. Ullman, 1981. Optimal wiring between rectangles, Proc. 13th STOC Symp., 312-317.

S. Gao and M. Kaufmann, 1987. Channel routing of multiterminal nets, Proc. 28th FOCS Symp., 316-325.

M. C. Golumbic, 1980. Algorithmic graph theory and perfect graphs, Academic Press, New York.

D. S. Johnson, 1982 and 1984. The NP-Completeness column: an ongoing guide, Journal of Algorithms, 3, 381-395 and 5, 1147-160, respectively.

B. W. Kernighan, D. G. Schweikert and G. Persky, 1973. An optimum channel-routing algorithm for polycell layouts of integrated circuits, Proc. 10th Design Automaation Workshop, 50-59.

A. S. LaPaugh, 1980. A polynomial time algorithm for optimal routing around a rectangle, Proc. 21st FOCS Symp., 282-293.

M. Marek-Sadowska and E. Kuh, 1983. Geneeral channel-routing algorithm, Proc. IEE (GB), 130, G, 3, 83-88.

A. Recski, 1989. 2-layer routing of dense bipartite specifications with vertex-disjoint paths and via-holes, Working paper OR572, University of Bonn, Institute for Operations Research.

T. G. Szymanski, 1982. Dogleg channel routing is NP-complete, unpublished manuscript, quoted in (Johnson, 1982).

T. Yoshimura and E. Kuh, 1982. Efficient Algorithms for Channel Routing, IEEE Trans. CAD-1, 25-35.

Perfection, Parity, Planarity, and Packing Paths

B. Reed, University of Waterloo.

Abstract

A hole is a chordless cycle with at least four vertices. We discuss some relationships between the holes of a graph and its chromatic number. In particular, we discuss triangulated graphs, perfect graphs, and a new analogue of perfect graphs: w-perfect graphs. This motivates our survey of a number of recent (and not so recent) results on the complexity of finding certain kinds of holes in a given graph. The paper also contains a similar discussion on induced odd length paths.

0. Introduction

By a **hole** , we mean an induced cycle of length at least four. We want to discuss some relationships between colourings, cliques, vertex degrees and holes. For a very brief instant, we ignore holes and consider colourings, cliques and degrees. A **colouring** of a graph G is an assignment of positive integers to the vertices of G so that no two adjacent vertices receive the same colour. $\chi(G)$, the chromatic number of G, is the minimum number of colours needed to colour G. A **clique** in G is a set of pairwise adjacent vertices of G. We use $\omega(G)$ to denote the size of the largest clique in G. We use $\delta(G)$ and $\Delta(G)$ respectively to denote the smallest and largest vertex degree in G. Clearly, $\chi(G)\geq\omega(G)$ and $\chi(G)\leq\Delta(G)+1$. Actually, letting $w(G)=\max\{\delta(H)+1|H\subseteq G\}$ we claim that $\chi(G)\leq w(G)$. To see this, let $v_1<v_2<\cdots<v_n$ be an ordering obtained by the following algorithm:

(1) set $i=n$, $F_n=V$

(2) let v_i be a minimum degree vertex of F_i and set $F_{i-1}=F_i-v_i$

(3) if $i=0$ stop, otherwise return to step 2.

Now colour G by colouring v_i with the smallest integer not used by any of its neighbours in F_i. The claim follows.

A graph without holes is called **triangulated** (or **chordal** or **rigid circuit**). Dirac [11] showed that any triangulated graph contains a vertex whose neighbourhood is a clique. It follows that, if G is triangulated then $\omega(G)=w(G)$. By the remarks above, if G is triangulated then $\omega(G)=\chi(G)=w(G)$. In fact, we see that G is triangulated if and only if for every vertex induced subgraph F of G we have $\chi(F)=\omega(F)=w(F)$. Furthermore, the colouring procedure discussed above optimally colours a triangulated graph.

In 1961, Berge [1] defined the class of perfect graphs. A graph G is **perfect** if for every induced subgraph F of G we have $\chi(F)=\omega(F)$. Grotschel, Lovasz and Schrijver [16] have shown that perfect graphs can be optimally coloured in polynomial time. It turns out that odd holes are closely related to perfection.

In 1989, Gasparian, Markossian, and Reed [24] defined the class of **w-perfect graphs**. A graph G is w-perfect if for all induced subgraphs of G, $\chi(F)=w(F)$. It turns out that even holes are closely related to w-perfection.

In section 1, we discuss the relationship between odd holes and perfect graphs. This motivates our discussion of the complexity of finding an odd hole in a graph. In section 2, we discuss the relationship between even holes and w-perfect graphs. This motivates our disussion of the complexity of finding an even hole in a graph. In section 3, we show that the perfection of a graph is closely related to the induced odd paths in the graph. This motivates our discussion of the complexity of finding an odd induced path between two specified vertices of a graph. In section 4, we discuss the complexity of finding holes through two specified vertices of a graph.

In what follows, H_i is the hole on i vertices, the length of an induced path is the number of edges it contains and P_i is the induced path of length $i-1$. We say x **sees** y if xy is an edge of G. A graph G is **Berge** if neither G nor $\overline{G}$ contain an odd hole.

1. Odd Holes and Perfect Graphs

Recall that G is perfect if for every induced subgragh F of G: $\chi(F)=\omega(F)$. If H is an odd hole then $\chi(H)=3$ and $\omega(H)=2$. Thus,

(1.1) No perfect graph contains an odd hole.

When he defined perfect graphs, Berge also put forward the following two conjectures :

Strong Perfect Graph Conjecture (SPGC): A graph G is perfect if and only if neither G nor $\overline{G}$ contains an odd hole.

Weak Perfect Graph Conjecture (WPGC): G is perfect if and only if $\overline{G}$ is.

The WPGC was proved in '71 by Lovasz [22]. The SPGC remains open. In light of Lovasz's result we can restate the SPGC as follows:

(1.2) G and $\overline{G}$ are perfect if and only if neither G nor $\overline{G}$ contains an odd hole (note that neccesity here is trivial it is sufficency which is difficult).

Two books [2,15] and hundreds of papers have been written about perfect graphs. Most of this flurry of activity was prompted by the above two conjectures. Another major motivating factor was the beautiful structural result obtained by Lovasz in his proof of the WPGC and elucidated independently by Chvatal [6] and Fulkerson [14]. Eventually, Grotschel, Lovasz, Schrijver[16] used this result and the ellipsoid method to prove:

(1.3) For a perfect graph G, $\chi(G)$ can be found in polynomial time.

Many of the papers on perfect graphs highlight relationships between perfect graphs and odd holes. Tucker [34] proved that a planar graph is perfect if and only if it contains no odd hole (Note that this is consistent with 1.2 since $H_5=\overline{H_5}$ and for $k\geq 3$, H_{2k+1} is not planar). Meyniel [26] and Markossian & Kerpetjan [23] independently proved:

(1.4) If G contains no odd hole and no odd cycle with precisely one chord then G is perfect (These graphs are called Meyniel).

Tucker [35] showed that if G_1 and G_2 are perfect graphs, S_1 in G_1 and S_2 in G_2 are stable sets with $|S_1|=|S_2|$, and G' is a graph obtained from G_1 and G_2 by identifying S_1 with S_2 then G' is perfect if and only if it contains no odd hole.

I hope that the above results give some feeling for the interplay between graphs with no odd holes and perfect graphs. We turn now to the complexity of finding odd holes. In particular, we are interested in the following question:

(1.5) Given a graph, determine if it contains an odd hole.

Although, the complexity of (1.5) is presently unknown, there are a number of partial results. We remark that if we could solve (1.5) and (1.2) were true then we could recognize perfect graphs in polynomial time (At the moment, we know only that perfect graph recognition is in Co-NP). Hsu [21] gave a polynomial-time recognition algorithm for planar perfect graphs. Since a planar graph is perfect if and only if it contains no odd hole, we obtain:

(1.6) Given a planar graph we can determine if it contains an odd hole in polynomial time.

Furthermore, Burlet and Fonlupt [5] have shown:

(1.7) Given a graph, we can determine if G contains an odd hole or an odd cycle with exactly one chord in polynomial time.

In contrast to (1.6) and (1.7), David Shmoys [32] has shown:

(1.8) Given a graph G and an integer k determining if G contains an odd hole of length at least k is NP complete.

2. Even Holes and W-perfect Graphs

Recall that a graph G is w-perfect if for each vertex induced subgraph F of G, $\chi(F)=w(F)$. If H is an even hole then $\chi(H)=2$ and $w(H)=3$. Thus,

(2.1) No w-perfect graph contains an even hole. (To see that the converse is false consider the graph of Fig. 1).

W-perfect graphs were defined by Gasparian, Markossian, and Reed [24]. We noted that the complement of a w-perfect graph need not be w-perfect, consider for example C_4 and $\overline{C_4}$. However, we did show that:

(2.2) G and $\overline{G}$ are w-perfect if and only if neither G nor $\overline{G}$ contains an even hole.

This result follows from the following characterization:

If neither G nor $\overline{G}$ contains an even hole then either:

(i) G consists of a clique C and stable set S, or

(ii) G consists of an H_5, a clique C and stable set S such that each vertex of the H_5 sees all of C and none of S.

The graphs which satisfy (i) are called split graphs and have been studied by Foldes and Hammer [12].

We note that w-perfect graphs are naturally of algorithmic interest because the colouring algorithm discussed in the introduction shows that:

(2.3) A w-perfect graph can be optimally coloured in polynomial-time.

Gasparian, Markossian, and Reed also showed :

(2.4) If G contains no even hole and no even cycle with exactly one chord then G is w-perfect.

This result followed from the following characterization:

If G contains no even hole and no even cycle with exactly one chord then each 2-connected component of G is a clique or an odd hole.

We actually obtained a strengthening of (2.4). To wit,

(2.4)' A **short chorded cycle** is a cycle with precisely one chord which forms a triangle with two edges of the hole. If G contains no even hole and no short-chorded even cycle then G is w-perfect.

Note that forbidding odd holes and odd cycles with one chord is equivalent to forbidding odd holes and short-chorded odd cycles. However, there are even cycles with exactly one chord which is not short, see for example Fig. 2. Thus (2.4)' is strictly stronger than (2.4). To prove (2.4)', we used the following lemma:

If G contains no even hole and no short chorded even cycle then either G contains a simplicial vertex or $\chi(G)=3$ and G contains a vertex of degree 2.

Finally we related even holes to w-perfect graphs once more by showing:

If G contains no even hole then $\chi(G) \geq w(G)/2$.

We turn now to the complexity of finding even holes. In particular, we are interested in the following question:

(2.5) Given a graph, determine if it contains an even hole.

The complexity of this question is unknown. However, a linear-time algorithm to determine if either G or $\overline{G}$ contains an even hole can easily be developed from our structural characterization of such graphs. We now mention a few other results related to (2.5). Oscar Porto and myself [29] have shown:

(2.6) There is a linear-time algorithm to determine if a planar graph contains an even hole.

Also, Gasparian, Markossian, and Reed have shown:

(2.7) There is a polynomial-time algorithm which given a graph G, determines if G contains an even hole or a short-chorded even cycle in polynomial time.

3. Odd Paths and Even Pairs

Two vertices x and y of a graph G form an **even pair** if there is no odd induced path between them. We shall see that the even pairs in a graph can be used to help colour it. To begin, we make a definition and an observation.

Definition: Let G be a graph. Let x and y be non-adjacent vertices of G. The graph obtained by **contracting** G on $\{x,y\}$ has vertex set $V-x-y+x'$ and edge set $E(G-x-y)+\{vx' \mid v \in G-x-y$ and v sees one of x or y in $G\}$.

Observation: If $\{x,y\}$ is an even pair of G and G' is obtained by contracting G on $\{x,y\}$ then $\chi(G')=\chi(G)$ and $\omega(G') = \omega(G)$.

Proof: Clearly $\chi(G')\geq\chi(G)$ and $\omega(G')\geq\omega(G)$ as this is true for any pair $\{x,y\}$. Now, $\omega(G')\leq\omega(G)$ because there is no induced path of length three between x and y. Since there is no odd induced path from x to y, there is a $\chi(G)$ colouring of G in which x and y have the same colour (if x and y have different colours, we swap these colours on the component of the graph induced by the vertices of these two colours which contains x). It follows that $\chi(G')\leq\chi(G)$.

A **contraction sequence** for a graph G is a sequence $G_0=G,G_1,\ldots,G_k$ such that G_k is a clique and for $1\leq i\leq k$, G_i is obtained from G_{i-1} via contraction on an even pair $\{x_i,y_i\}$. By our observation, if $G_0,\cdots,G_k$ is a contraction sequence for G then $\omega(G)=\chi(G)=|G_k|$. Actually, by working backwards from G_k we can find an optimal colouring and maximal clique in G_0. To obtain a colouring of G_{i-1} from G_i we simply give x_i and y_i the same colour as x_i'. To obtain a maximal clique of G_{i-1} from a maximal clique in G_i we simply replace x_i' by one of x_i or y_i if necessary. Thus, if we can find a contraction sequence for a graph G in polynomial time then we can optimally colour the graph in polynomial time.

It turns out that we can do this for a number of classes of perfect graphs. Consider, for example, the class of weakly triangulated graphs. A graph G is **weakly triangulated** if neither G nor $\overline{G}$ contains a hole of length at least 5. Hayward [18] showed that these graphs were perfect. Hayward, Hoang and Maffray [19] showed that every weakly triangulated graph is either a clique or contains a **two-pair**, that is a pair of vertices $\{x,y\}$ such that every chordless path from x to y has length 2. They also showed that contracting a two-pair in a weakly triangulated graph produces a new weakly triangulated graph. Finally, they noted that we can check if a graph has a two-pair and find such a pair if it exists in polynomial time. Thus, for any weakly triangulated graph G, we can find a contraction sequence and therefore an optimal colouring and maximum clique of G in polynomial time. Furthermore, their result

gives a new proof that weakly triangulated graphs are perfect. Similarly, Hertz [20] proved that we can find a contraction sequence for any Meyniel graph in polynomial time. This gives an algotithm for colouring and finding the largest clique in these graphs. It also gives a new proof that these graphs are perfect.

We note that we have implicitly stated above that if $\mathcal{F}$ is an hereditary family of graphs (this means that if G is in $\mathcal{F}$ so are all its induced subgraphs) and we can find a contraction sequence for every graph in G then $\mathcal{F}$ is a family of perfect graphs. Actually, Meyniel [27], and Fonlupt & Uhry [13] had all already shown (independently) that no minimal imperfect graph contains an even pair (G is minimal imperfect if G is not perfect but all of its proper induced subgraphs are); (For a generalization, see Bertschi & Reed [3]). It follows that if $\mathcal{F}$ is an hereditary family of graphs each of which is a clique or contains an even pair then $\mathcal{F}$ is a family of perfect graphs.

Meyniel [26] calls a graph **Strict Quasi-Parity** if each of its subgraphs either is a clique or contains an even pair. A graph is **Quasi-Parity** if for each of its induced subgraphs F, either F is a clique or one of $F,\overline{F}$ contains an even pair. The remarks above imply directly that every SQP graph is perfect. Combined with Lovasz's result, they also imply that every QP graph is perfect.

These remarks and examples suggest techniques for proving the SPGC. We could, for example, try to show that every Berge graph is a QP graph. Unfortunately, this is false, as demonstrated by the graph F in Fig. 3 (this particular example comes from de Figueredo & Tardif [9]). However, it might be that there is some simple base class B of graphs which are known to be perfect and for which the following is true:

(3.1) For every Berge graph G, either G is in B or G has an even pair or $\overline{G}$ has an even pair.

This would imply the SPGC.

We note that any such B must contain the line graphs of bipartite graphs (since many of these have no even pair and no even pair in the complement), as well as the graph F. Note that by our observation, if G' is obtained from G by contracting on an even pair then G' is perfect if and only if G is. Thus, if we had:

(i) A result of the type described in (3.1) such that there is a polynomial-time recognition algorithm for B, and

(ii) A polynomial time algorithm which finds an even pair in a perfect graph if it contains one and which when presented with an imperfect graph either finds an even pair or shows that the graph is imperfect,

then we would have a polynomial-time recognition algorithm for perfect graphs. Unfortunately, Bienstock [4] has shown that determining if an arbitrary graph contains an even pair is NP-complete. However, I would like to suggest the following two conjectures:

Conjecture A: We can determine if vertices x and y of a perfect graph form an even pair in polynomial-time.

Conjecture B: We can determine if a given perfect graph contains an even pair and find one if it does in polynomial time.

As evidence for these conjectures I point to the results of Hayward, Hoang & Maffray [19], and Hertz [20] mentioned earlier. Also, Hsu [21] has shown that if G is a planar perfect graph and x and y are two vertices on the same face of G then we can determine if x and y form an even pair in polynomial-time (We simply add a vertex z and edges zx, zy and check if the resulting planar graph is perfect).

The above remarks show that not only do the odd holes of a graph affect its perfection, so do the odd paths. We close this section by mentioning a completely different result which illustrates the relationship between perfect graphs and odd holes. In 1982, V. Chvatal [7] proposed the following conjecture.

Semi-Strong Perfect Graph Conjecture (SSPGC): Let G and H be graphs on the same vertex set V, such that any four vertices in V form a P_4 in G if and only if they form a P_4 in H. Then G is perfect if and only if H is perfect (since $\overline{P_4}=P_4$ this implies the WPGC, it is also implied by the SPGC).

I proved the SSPGC in 1986 [30]. Thus, we see that the perfection of a graph depends only on the paths of length three in a graph. Two algorithmic questions related to this result are:

(3.2) Given two graphs G and H, is there a bijection f from $V(G)$ to $V(H)$ such that $\{a,b,c,d\}$ forms a P_4 in G if and only if $\{f(a),f(b),f(c),f(d)\}$ forms a P_4 in H,

(3.3) Given a set V and a set S of 4-element subsets of V, is there a graph G such that $\{a,b,c,d\}$ is a P_4 in G if and only if $\{a,b,c,d\}$ is in S.

I showed that (3.2) is (polynomially) equivalent to graph-isomorphism. The complexity of (3.3) is unknown. Hayward [17] has some results on the analagous questions for paths of length two.

4. Packing Induced Paths

In the last section, we discussed the complexity of determining whether or not there is an odd induced path between two given vertices of a graph. In this section, we discuss a number of related problems. They are all special cases of the following decision problem:

INDUCED PATH PACKING

Input: A graph G, two vertices s and t of G, and three integers k,e, and o.

Question: Can we find k vertex disjoint induced paths of G, such that there are at least e even paths, there are at least o odd paths, and there are no edges between the internal vertices of any two distinct paths.

We note that the problem of the last section is simply INDUCED PATH PACKING with $k=o=1$ and $e=0$. Determining if s and t lie on an induced cycle is simply INDUCED PATH PACKING with $k=2$ and $e=o=0$. Bienstock [4] has also shown that this problem is NP-complete. Determining if s and t lie on an odd hole is simply INDUCED PATH PACKING with $k=2$ and $e=o=1$. From Bienstock's results, we also obtain that this problem (as well as that of determining if s and t lie on an even hole) is NP-complete.

In contrast, McDiarmid, Reed, Schrijver, and Shepherd [25] have shown that if we insist that $e=o=0$ and that G is planar, then we can solve INDUCED PATH PACKING in polynomial time. Furthermore, Schrijver [32] has shown that for any fixed integer n and surface Σ there is a polynomial time algorithm to solve induced path packing instances such that G is embeddable on Σ, $k \leq n$ and $e=o=0$. It would be of interest to know what other restrictions of INDUCED PATH PACKING can be solved in polynomial time. In particular, what is the complexity of INDUCED PATH PACKING if G is perfect.

5. Possibly Pertinent Problems

In this section, we shall mention some more open problems which are related to the problems and results we have discussed.

(1) As stated in Section 2, if a graph has no even hole then $\chi(G) \geq \omega(G)/2$. I asked if there is a function f such that if G contains no odd hole, then $\chi(G) \leq f(\omega(G))$. We do not know of such a function even if G is a Berge graph. Note that for any k there are graphs with no cycle of length less than 6 (and hence no complement of an odd hole) and $\chi(G)=k$.

(2) Many of the problems we have discussed are easier if we restrict our attention to planar graphs. In fact, some of these problems can be solved in polynomial time if we restrict ourselves to graphs embedded on any fixed surface (determining if two vertices lie on an induced cycle [32], determining if a graph contains an even hole [31]). It would be interesting to know which of the other problems we have discussed can be solved quickly on graphs embedded on a fixed surface.

(3) De Werra [10] has conjectured that if a graph contains an even pair then it contains an even pair {x,y} such that x and y have a common neighbour.

(4) In attempting to find even pairs, we could restrict our attention to special classes of perfect graphs. In particular, it would be of interest to show that we can find an even pair in any SQP graph, which is not a clique, in polynomial time. Even better would be to show that we can find an even pair in these graphs such that contracting on the pair gives a new SQP graph. Another interesting class of graphs are the perfectly orderable graphs, defined by Chvatal [8]. We know that any such graph contains an even pair. It would be nice to develop a polynomial-time algorithm to find contraction sequences for graphs in this class.

(5) It would be nice to obtain a good characterization of w-perfect graphs. Even the restricted class of planar w-perfect graphs seems interesting.

(6) Olariu [28] showed that no minimal imperfect graph contains a pair of vertices $\{x,y\}$ such that every induced path between x and y has length three. Two non-adjacent vertices of G form an **odd pair** if there is no even induced path between them. Subsequent to Olariu's result, many people asked:

Is it true that no minimal imperfect graph contains an odd pair.

Note that this statement is implied by the SPGC.

References

[1] Berge C., "Farbung von Graphen, deren samtliche bzw. deren ungerade Kreise starr sind.", Wiss. Z. Martin Luther-Univ., Halle-Wittenberg, Math.-Natur. Reihe, 114, 1961.

[2] Berge C. and Chvátal V., "Topics on perfect graphs", North Holland, Amsterdam, 1984.

[3] Bertschi M. and Reed B., "A note on even pairs", Discrete Mathematics, Vol. 65, p. 317, 1987.

[4] Bienstock D., "On the complexity of testing for odd induced cycles and odd induced paths", to appear in Discrete Mathematics.

[5] Burlet M. and Fonlupt J., "Polynomial algorithm to recognize a Meyniel graph", pp. 225-252 in [2].

[6] Chvátal V., "On certain polytopes associated with graphs", J.C.T.(B), Vo. 18, pp. 138-154, 1975.

[7] Chvátal V., "A semi-strong perfect graph conjecture", pp. 279-280 in [2].

[8] Chvátal V., "Perfectly ordered graphs", pp. 63-68 in [2].

[9] de Figueredo C. and Tardif V., "Some problems on perfect graphs", University of Waterloo, Department of Combinatorics and Optimization, Research Report 89-31, 1989.

[10] De Werra D., private communication.

[11] Dirac G.A., "On rigid circuit graphs", Abh. Math. Sem. Univ. Hamburg 25, pp. 71-76, 1961.

[12] Foldes S. and Hammer P., "Split graphs", Proc. 8th Southeastern conference on Combinatorics, Graph Theory and Computing, pp. 311-315, 1977.

[13] Fonlupt J. and Uhry J.P., "Transformations which preserve perfectness and H-perfectness of graphs", Annals of Discrete Mathematics, Vol. 16, pp. 83-95, 1982.

[14] Fulkerson D., "Anti-blocking polyhedra", J.C.T.(B), Vol. 12, pp. 50-71, 1972.

[15] Golumbic M., "Algorithmic graph theory and perfect graphs", Academic Press, New York, 1980.

[16] Grotschel M., Lovasz L. and Schrijver A., "Polynomial algorithms for perfect graphs " in (C. Berge, V. Chvátal, "Topics on perfect graphs"), North Holland, Amsterdam, pp. 325-356, 1984.

[17] Hayward R., "Recognizing P_3-structure", submitted.

[18] Hayward R., "Weakly triangulated graphs", J.C.T.(B), Vol. 39, pp. 200-208, 1985.

[19] Hayward R., Hoang C.T. and Maffray F., "Optimizing weakly triangulated graphs", submitted.

[20] Hertz A., "Fast colouring of Meyniel graphs", submitted.

[21] Hsu W.L., "Recognizing planar perfect graphs", Journal of the A.C.M., Vol. 34, pp. 255-288, 1987.

[22] Lovasz L., "Normal hypergraphs and the perfect graph conjecture", Discrete Math, Vol. 2, pp. 253-267, 1972.

[23] Markossian S. and Kerpetjan I., "Perfect graphs" (in Russian), Akad. Nauk. Armjan. SSR DOKL 63, pp. 292-296, 1976.

[24] Markossian S., Gasparian G. and Reed B., "W-Perfect graphs", submitted to J.C.T.(B).

[25] McDiarmid C., Reed B.A., Schrijver A. and Shepherd B., "Induced holes in planar graphs", manuscript.

[26] Meyniel H., "On the perfect graph conjecture", Discrete Math, 16, pp. 253-267, 1976.

[27] Meyniel H., "A new property of critical imperfect graphs and some consequences", Europ. J. Combinatorics, Vol. 8, pp. 313-316, 1987.

[28] Olariu S., "No antitwins in minimal imperfect graphs", J.C.T.(B), Vol. 45, pp. 255-257, 1988.

[29] Porto O. and Reed B., "Even holes in planar graphs", manuscript.

[30] Reed B., "A semi-strong perfect graph theorem", J.C.T.(B), Vol. 43, pp. 223-240, 1987.

[31] Reed B., "Finding even holes in an embedding, in preparation.

[32] Schrijver A., private communication.

[33] Shmoys D., private communication.

[34] Tucker, A., "The strong perfect graph conjecture for planar graphs", Can. Journal of Mathematics, Vol. 25, pp. 103-114, 1973.

[35] Tucker A., "Colouring graphs with stable sets", J.C.T.(B), Vol. 34, pp. 258-267, 1983.

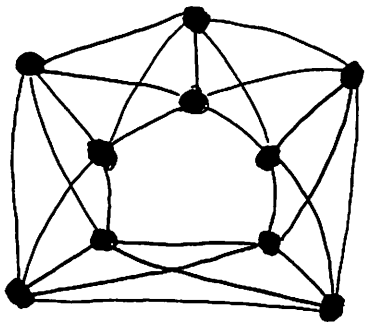


Figure 1

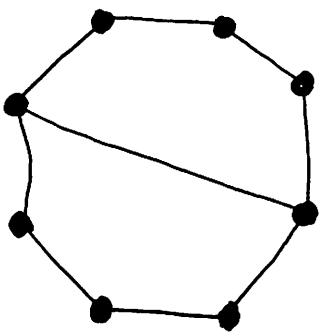


Figure 2

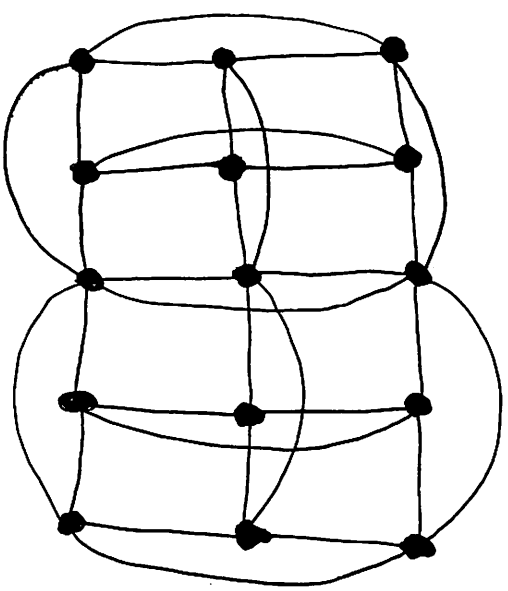


Figure 3

Max-Balanced Flows

Michael H. Schneider*
Department of Mathematical Sciences
Johns Hopkins University
Baltimore, Maryland 21218

April 1, 1990

This paper contains a survey of work that is joint with:

Mark Hartmann
Department of Operations Research
University of North Carolina
Chapel Hill, North Carolina 27599

Hans Schneider
Department of Mathematics
University of Wisconsin
Madison, Wisconsin 53706

See, references:

[4] Mark Hartmann and Michael H. Schneider. "An analogue of Hoffman's circulation theorem for max-balanced flows," 1989.

[8] Hans Schneider and Michael H. Schneider. "Max-balancing weighted directed graphs and matrix scaling," 1989.

[9] Hans Schneider and Michael H. Schneider. "Towers and cycle covers for max-balanced graphs", 1990.

Abstract

Let $G = (V, A)$ be a strongly-connected digraph, and let f be an weight-function for G (i.e. an arbitrary real-valued function defined on the arcs A). The function f is called a *max-balanced flow* (or simply *max-balanced*) if for every cut W, $\emptyset \subset W \subset V$, the maximum weight over arcs directed out of W equals the maximum weight over arcs directed into W. In [8] we were motivated by problems in matrix scaling to ask the following question, which we call MAX-BALANCING: Given an arbitrary weight function f, can you find a potential p such that the weight function $f_a^p = p_v + f_a - p_v$ for $a = (u, v) \in A$ is max-balanced? We showed that such a potential is unique (up to an additive constant) and described an efficient algorithm based on computing maximum cycle-means. We give an informal proof of this uniqueness result and an informal description of the algorithm

Next, we state two characterizations of max-balanced flows using f-cycle covers for G and bottleneck paths for G. Finally, for given weight functions $l \leq u$ we describe an analogue of

*Research supported in part by NSF grant ECS 87-18971.

Hoffman's circulation theorem for the existence of a max-balanced flow f satisfying $l \leq f \leq u$. We describe some results about the structure of the set of all max-balanced flow satisfying $l \leq x \leq u$, and compare this set with the set of all circulations satisfying $l \leq x \leq u$. In particular, we show that a max-balance flow satisfying $l \leq f \leq u$ can be computed efficiently, but show that the problem of minimizing a linear function over all such max-balanced flows is NP-hard.

1 Introduction

Let $G = (V, A)$ be a digraph with *vertex set* V and *arc set* $A \subseteq V \times V$. We use the notation $a = (u, v)$ to denote an arc $a \in A$ directed from vertex u to vertex v. We use the symbols $\subset$ and $\subseteq$ to denote, respectively, strict and weak containment. A *cut* of G is a nontrivial subset W of the vertices (i.e. $\emptyset \subset W \subset V$) . Given a cut W of G, we define the set of arcs *leaving* W and the set of arcs *entering* W, written $\delta^+(W)$ and $\delta^-(W)$, respectively, by

$$\begin{aligned} \delta^+(W) &= \{a = (u, v) \in A \mid u \in W, \text{ and } v \in V \backslash W\}, \quad \text{and} \\ \delta^-(W) &= \{a = (u, v) \in A \mid u \in V \backslash W, \text{ and } v \in W\}. \end{aligned}$$

For a finite set S, we will use $\Re^S$ to denote the set of all real-valued functions with domain S. A *weight function* for a digraph $G = (V, A)$ is a function $f \in \Re^A$. We will use f_a for $a \in A$ to denote the weight of arc a. Given a cut W for G, we say that a weight-function f is *max-balanced at* W if

$$\max_{a \in \delta^+(W)} f_a = \max_{a \in \delta^-(W)} f_a.$$

We say that f is a *max-balanced flow for* G (or simply *max-balanced*) if it is max-balanced at every cut W.

A *potential* for a digraph $G = (V, A)$ is a function $p \in \Re^V$. Given a weight function f and a potential p for G, we define the *reduced weights of* f *with respect to* p to be the weight function f^p defined by

$$f^p_a = p_u + f_a - p_v \quad \text{for } a = (u, v) \in A. \tag{1}$$

We note that reduced weights with respect to potentials arise throughout network optimization (see, for example, [6]).

Throughout this paper *paths* and *cycles* of a digraph G will be simple and directed. We will identify a cycle C, its underlying vertex set, and its underlying arc set. For example, $V \backslash C$ refers to the vertices of V not contained in C. Also, we use $|C|$ to denote the number of arcs of C.

Let f be a weight function for G, and let C be a cycle of G. We define the *mean of* C, written $\bar{f}(C)$, by

$$\bar{f}(C) = \frac{1}{|C|} \sum_{a \in C} f_a.$$

We define the *maximum cycle mean of* G *with respect to* f, by

$$\text{mcm}((G, f)) = \max\{\bar{f}(C) \mid C \text{ is a cycle for } G\}$$

For notational convenience, we will write $\text{mcm}(f)$ and omit the dependence on G. A cycle C of G is a *maximum mean cycle of* G if $\bar{f}(C) = \text{mcm}(f)$.

Next, we define informally the operation of contraction with respect to a cycle. For a cycle C of the digraph $G = (V, A)$, we define the *contraction of* G *with respect to* C, written G/C, to be the digraph derived from G by shrinking the subgraph induced by C to a point and identifying all resulting parallel arcs. Thus, the vertices of G/C are the vertices in $V \backslash C$ together with a distinguished vertex C, which we refer to as a *pseudo-vertex*. For $u, v \in V \backslash C$, the digraph G/C

contains arc (u,v) whenever $(u,v) \in A$. Also, G/C contains arc (C,v) whenever $(u,v) \in A$ for some vertex $u \in C$. Similarly, G/C contains arc (u,C) whenever $(u,v) \in A$ for some vertex $v \in C$.

We define the weight function f/C for G/C by *max-projecting* f onto G/C. That is, if $a = (u,v) \in A$ where $u,v \in V\backslash C$, then $(f/C)_a = f_a$. For the *pseudo-vertex* C and $v \in V\backslash C$ we define

$$\begin{aligned} (f/C)_{(C,v)} &= \max\{f_a \mid a = (u,v) \in A, u \in C\} \quad \text{and} \\ (f/C)_{(v,C)} &= \max\{f_a \mid a = (v,u) \in A, u \in C\}. \end{aligned} \tag{2}$$

(Of course, arcs (C,v) and (v,C) are in G/C if and only the respective maxima in (2) are defined over nonempty sets.)

2 The Max-Balancing Problem

Consider the following problem:

> MAX-BALANCING: Given a strongly-connected digraph G and a weight function f for G, find a potential p such that f^p is max-balanced.

Schneider and Schneider proved the following result in [8]:

Theorem 1 *Let $G = (V,A)$ be a strongly-connected digraph, and let f be a weight function for G. Then there exists a unique (up to additive constant) potential p for G such that f^p is max-balanced.*

It is straightforward to show that in Theorem 1, if p and q are two such potentials, then $p - q$ is a constant. We proved existence constructive by exhibiting an algorithm for computing p that requires solving at most $|V|$ maximum cycle mean problems. Next, we give an informal summary of the algorithm in [8].

We used the following duality result of Engle and Schneider [3]:

Theorem 2 *Let $G = (V,A)$ be a strongly-connected digraph, and let f be a weight function for G. Then*

$$\operatorname{mcm}(f) = \min_{p \in \mathbb{R}^V} \left\{ \max_{\substack{a \in A \\ a=(u,v)}} \{p_u + f_a - p_v\} \right\}.$$

Since $\bar{f}^p(C) = \bar{f}(C)$ for any potential p and cycle C, it follows directly from properties of maxima and averages that if C^* is a maximum-mean cycle then for any potential p there exists some $a \in C^*$ such that $f^p_a \geq \operatorname{mcm}(f)$. It follows that

$$\operatorname{mcm}(f) \leq \min_{p \in \mathbb{R}^V} \left\{ \max_{\substack{a \in A \\ a=(u,v)}} \{p_u + f_a - p_v\} \right\}.$$

To complete the proof of Theorem 2, define p_v to be the maximum weight over all paths ending at v with respect to the weight function $f - \operatorname{mcm}(f)$ (i.e. the weight function defined by decreasing f_a by $\operatorname{mcm}(f)$ for $a \in A$). It follows directly from the definition of p that

$$p_u + f_a - \operatorname{mcm}(f) \leq p_v \quad \text{for } a = (u,v) \in A,$$

and Theorem 2 follows. Theorem 2 can also be proved using linear programming duality. The potential p can be computed in $O(|V||A|)$ using, for example, an algorithm of Karp as described in [5].

We now describe informally our method for computing for a given weight function f the potential p for which f^p is max-balanced. First, we compute a maximum-mean cycle C for G and a potential q such that $f_a^q \leq \mathrm{mcm}(f)$ for all $a \in A$. We then shrink the cycle C to a point and max-project the function f^q onto the resulting digraph G/C. We repeat this operation until the resulting digraph is a singleton. Then we define the potential p at vertex v by adding up the potentials q computed at each iteration evaluated at the pseudo-vertex containing v.

The proof that f^p is max-balanced uses the following two results:

(i) Let λ_i be the maximum cycle-mean computed at iteration i. Then

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m.$$

(ii) For each $a = (u, v) \in A$ the weight f_a^p is the weight on a at the time u and v are contracted into the same pseudo-vertex.

Now to prove that f^p is max-balanced, let W be any cut of G and *run the algorithm* until the computed maximum-mean cycle C_j at the j-th iteration intersects $\delta^+(W)$, and therefore also $\delta^-(W)$. There must be such a j since the algorithm terminates when the contracted digraph is a singleton. Let λ_j be the mean of C_j. It follows from Theorem 2 that at iteration j, the weights satisfy the max-balance condition. By observation (ii) the weights on the arcs of C_j are unchanged at subsequent iterations. By observation (i) the remaining arcs of $\delta^+(W) \cup \delta^-(W)$ never exceed λ_j. Therefore, the balanced condition remains satisfied at subsequent iterations.

3 Matrix Scaling

In this section we describe the connection between max-balancing and matrix scaling—the original motivation for studying MAX-BALANCING. A square nonnegative matrix B is called *balanced* if

$$\sum_j b_{ij} = \sum_j b_{ji} \quad \text{for each } i. \tag{3}$$

That is, for each index i the sum of the entries in the i-th row equals the sum of the entries in the i-th column. MAX-BALANCING is an analogue of the following matrix scaling problem:

> MATRIX SCALING: Given a square nonnegative matrix B, find a positive diagonal matrix D such that DBD^{-1} is balanced.

For a discussion of this problem and related generalizations, see [10,11] and the references therein.

For an $n \times n$ nonnegative matrix B, the *digraph of* B is the digraph (V', A') where

$$V' = \{1, 2, \ldots, n\},$$

and

$$A' = \{(i, j) \in V' \times V' \mid a_{ij} > 0\}.$$

We define the weight function corresponding to B by $f_{ij} = a_{ij}$ for $(i, j) \in A'$. Under the correspondence between square nonnegative matrices and ordered pairs (G, f) of digraphs and positive weight functions, a matrix B is balanced if and only if the weight function f corresponding to B is a circulation, that is, if and only if f satisfies

$$\sum_{a \in \delta^+(W)} f_a = \sum_{a \in \delta^-(W)} f_a,$$

for every cut W for G.

Thus, MATRIX SCALING is equivalent to the digraph problem:

DIGRAPH SCALING: Given a digraph $G = (V, A)$ and a positive weight function f, find a positive potential p such that the weight function g defined by $g_a = p_u f_a p_v^{-1}$ for $a = (u, v) \in A$ is a circulation for G.

A sufficient condition for the existence of a solution to DIGRAPH SCALING is that G is strongly-connected. See [1] for necessary and sufficient conditions.

Next, we observe that for $0 < \rho < \infty$, we can generalize DIGRAPH SCALING by requiring that the weight function g in the statement of the problem satisfy

$$\left\{ \sum_{a \in \delta^+(W)} (g_a)^\rho \right\}^{1/\rho} = \left\{ \sum_{a \in \delta^-(W)} (g_a)^\rho \right\}^{1/\rho}, \tag{4}$$

for each cut W for G. It is straightforward to show that the cases of $0 < \rho < \infty$ can be reduced to DIGRAPH SCALING (which is the case of $\rho = 1$) by considering the ρ-th powers of the weight function f (i.e. the weight function whose value at arc a is $(f_a)^\rho$). In the limiting case of $\rho = \infty$, (4) converges to

$$\max_{a \in \delta^+(W)} g_a = \max_{a \in \delta^-(W)} g_a. \tag{5}$$

In the case of $\rho = +\infty$, the resulting variant of DIGRAPH-SCALING is transformed into MAX-BALANCING by making the change of variable $p'_v = \ln p_v$ for $v \in V$ and $f'_a = \ln f_a$ for $a \in A$ and taking the logarithm of each side of (5). Thus, we are motivated to study MAX-BALANCING.

4 Characterizations of Max-Balanced Flows

In this section, we describe two characterization of max-balanced flows. Further characterization are contained in [4,7,9].

4.1 Bottleneck Path Characterization

Let $G = (V, A)$ be a digraph, and let f be a weight function for G. We will use $Path(s, t)$ to denote the see of all directed paths from s to t. For vertices s and t of G, an (s, t) *bottleneck path with respect to* f is an (s, t) path Q such that

$$\min_{a \in Q} f_a = \max_{P \in Path(s,t)} \left\{ \min_{a \in P} f_a \right\}.$$

That is, a bottleneck path is a path for which the minimum weight is maximized.

Hartmann and Schneider [4] showed that:

Theorem 3 *A weight function f for the digraph G is max-balanced if and only if for every pair of vertices s and t,*

$$\max_{P \in Path(s,t)} \left\{ \min_{a \in P} f_a \right\} = \max_{P \in Path(t,s)} \left\{ \min_{a \in P} f_a \right\} \tag{6}$$

Next we describe a setting for a max-balanced flow problem based on Theorem 3. Suppose that the digraph $G = (V, A)$ models a physical transportation network in which the vertices and arcs represent, respectively, intersections and streets in an urban area. Suppose that the weight f_a of arc a represents the maximum height vehicle that can travel on arc a as a result, for example, of an overpass, walkway, or hanging sign. In this physical setting, the weight function f is max-balanced if and only if whenever a vehicle can travel from origin s to destination t then there exists an unobstructed return path. Thus, when a city planner is designing the specified height clearances, he should include among other design considerations the requirement that the resulting function f is max-balanced.

4.2 Cycle Cover Characterization

Let $G = (V, A)$ be a digraph, and let f be a weight function for G. A set of cycles of G indexed on the arcs A, $\{C_b \mid b \in A\}$, is an *f-cycle cover for G* if for each $b \in A$,

(i) $b \in C_b$, and

(ii) $f_b \leq f_a$ for $a \in C_b$.

It follows directly that G has an f-cycle cover if and only if every arc $b \in A$ is contained in some cycle for which it is the minimum weight arc.

Schneider and Schneider in [9, Corollary 8] proved the following characterization of max-balanced flows:

Theorem 4 *Let $G = (V, A)$ be a strongly-connected digraph, and let f be a weight function for G. Then f is max-balanced if and only if there exists an f-cycle cover for G.*

Theorem 4 is an analogue for max-balanced flows of the well-known result for circulations that a weight function is a circulation if and only if it can be decomposed into a sum of flows around cycles. Specifically, let $\mathcal{C} = \{C_b \mid b \in A\}$ be an f-cycle cover for G. For $C \in \mathcal{C}$, where $C = C_b$ for some $b \in A$, define

$$\omega(C) = f_b.$$

Then it follows directly from condition (ii) in the definition of an f-cycle cover that

$$f_a = \max\{\omega(C) \mid a \in C \in \mathcal{C}\}. \tag{7}$$

Moreover, it is straightforward to show that if $\mathcal{C}$ is any set of cycles of G and $\omega(C)$ for $C \in \mathcal{C}$ is an assignment of real numbers to the elements of $\mathcal{C}$, then the weight-function f defined by (7) is max-balanced.

5 Max-Circulation Results

A max-balanced flow is the analogue of a circulation in which the summation operator is replaced by a maximization operator. For given weight functions $l \leq u$, the problem of studying the set of circulations f satisfying $l \leq f \leq u$ is a classical problem in the theory of network flows, and the seminal result is, perhaps, Hoffman's circulation theorem characterizing the existence of such a circulation. Thus, we are motivated to ask whether or not there is a characterization of the existence of a max-balanced flow f satisfying $l \leq f \leq u$. We describe an analogue of Hoffman's circulation theorem for max-balanced flows and discuss related structural results concerning the set of all max-balanced flow satisfying given lower and upper bounds.

Let l and u be given weight functions for the digraph $G = (V, A)$, and assume that $l_a \leq u_a$ for $a \in A$. We define the set of *feasible max-balanced flows*, written $\mathrm{mbf}(l, u)$, by

$$\mathrm{mbf}(l, u) = \{f \in \Re^A \mid f \text{ is max-balanced, and } l \leq f \leq u\}.$$

For vertices s and t of G, we use $cut(s, t)$ to denote the set of all cuts W of G such that $s \in W$ and $t \in V \backslash W$. Recall that $Path(s, t)$ denotes the set of all (directed) paths of G from vertex s to vertex t.

The following theorem, which was proved by Hartmann and Schneider [4], describes an analogue of Hoffman's circulation theorem for max-balanced flows.

Theorem 5 *Let $G = (V, A)$ be a digraph, and let $l, u \in \Re^A$ satisfy $l \leq u$. Then the following are equivalent:*

(i) *For each cut W of D, we have*

$$\max_{a\in\delta^+(W)} l_a \le \max_{a\in\delta^-(W)} u_a. \tag{8}$$

(ii) *For any vertices $s \neq t$ in D,*

$$\min_{W\in cut(s,t)} \left\{ \max_{a\in\delta^+(W)} l_a \right\} \le \min_{W\in cut(t,s)} \left\{ \max_{a\in\delta^+(W)} u_a \right\}. \tag{9}$$

(iii) *For any vertices $s \neq t$ in D,*

$$\max_{P\in Path(s,t)} \left\{ \min_{a\in P} l_a \right\} \le \max_{P\in Path(t,s)} \left\{ \min_{a\in P} u_a \right\}. \tag{10}$$

(iv) *For each (s,t) path P there exists a (t,s) path Q such that*

$$\min_{a\in P} l_a \le \min_{a\in Q} u_a.$$

(v) *For each $b \in A$, there exists a cycle C_b for G such that $b \in C_b$ and*

$$l_b \le u_a \quad \text{for } a \in C_b.$$

(vi) *The set* $\mathrm{mbf}(l,u)$ *is nonempty.*

Note that part (i) of Theorem 5 can be transformed into Hoffman's circulation conditions by replacing the maximization operators in (8) by summation operators. Also, the equivalence of part (ii) and (iii) can be proved using the duality theory for bottleneck paths and bottleneck cuts [2].

We observe that a given $f \in \Re^A$ is max-balanced if and only if the set $\mathrm{mbf}(f,f)$ is nonempty. It follows that any characterization for the nonemptiness of $\mathrm{mbf}(l,u)$ will reduce to a characterization for max-balanced flows by considering the characterization applied to the set $\mathrm{mbf}(f,f)$. In particular, in Theorem 5 for the case of $l = u = f$ for a given weight function f, part (iii) reduces to the bottleneck path characterization described in Section 4.1 and part (v) reduces to the cycle cover characterization described in Section 4.2.

Using a labeling algorithm, one can find an element f in $\mathrm{mbf}(l,u)$ or show that no such element exists. Furthermore, using the fact that the pointwise maximum of two max-balanced flows is max-balanced, it is straightforward to show that the set $\mathrm{mbf}(l,u)$ contains a unique maximal element under the usual coordinatewise partial order. This element can be computed in time $O(|V||A|)$ using an all-pairs bottleneck path algorithm. See Hartmann and Schneider [4] for further structural results concerning the set $\mathrm{mbf}(l,u)$.

Next, we consider some complexity results concerning max-balanced flows. For a digraph $G = (V,A)$, let M be the *vertex-arc* incidence matrix for the G, and let f be a weight function for G. Then it follows that

$$\{f^p \mid p \text{ is a potential for } G\} = \left\{ f + M^{\mathrm{T}} p \mid p \text{ is a potential for } G \right\}. \tag{11}$$

Thus, it follows from Theorem 1 that there is a unique element in the intersection of the set of max-balanced flows and the affine set (11). Moreover, the algorithm of [8] described informally in Section 2 can be used to compute this element in time $O(|V|^2|A|)$.

Next, we consider the problem of finding a max-balanced element in more general polyhedral sets. For example, the set $\mathrm{mbf}(l,u)$ is the intersection of the box $\{f \in \Re^A \mid l \le f \le u\}$ with the set of max-balanced flows, and the results of [4] described above show that an element of $\mathrm{mbf}(l,u)$ can be found in time $O(|V||A|)$. It follows directly from the part (v) of Theorem 5 that the decision

problem associated with the problem of minimizing a linear function $c^T x$ over the set $\text{mbf}(l, u)$ is in the class NP. We show, however, that a restricted version of this problem is NP-hard.

We say that a digraph $G = (V, A)$ is *max-balanced* if the weight function $f_a = 1$ for $a \in A$ is a max-balanced flow for G. It can be shown that a digraph G is max-balanced if and only if it is the union of its strongly connected components. Consider the problem:

> **MAX-BALANCED SUBGRAPH**: Given a digraph $G = (V, A)$, a subset $B \subseteq A$ and a positive integer $K \leq |A|-1$, is there a max-balanced subgraph of G that includes all of the arcs in B and contains no more than K arcs?

It was shown in [4] that the STEINER TREE problem can be reduced to MAX-BALANCED SUBGRAPH, thereby proving that this problem is NPcomplete. For an instance of MAX-BALANCED SUBGRAPH, define l, u, and c by

$$l_a = \begin{cases} 1 & \text{if } a \in B, \quad \text{and} \\ 0 & \text{if } a \in A \backslash B \end{cases}$$

and $u_a = c_a = 1$ for $a \in A$. It is straightforward to show that for these choices of l, u, and c, a solution for the problem of minimizing $c^T x$ over the set $\text{mbf}(l, u)$ provides an answer for MAX-BALANCED SUBGRAPH, and therefore it follows that the minimization problem is NP-hard.

For l and u as defined above, consider the constraints

$$\sum_{a \in A} x_a \leq K,$$

and

$$l \leq x \leq u.$$

Then MAX-BALANCED SUBGRAPH is equivalent to determining whether or not there exists a max-balanced flow satisfying these inequalities. It follows that for general inequality constraints $Mx \leq b$, the problem of determining whether or not the set $\{x \mid Mx \leq b\}$ contains a max-balanced element is NP-hard.

References

[1] B. Curtis Eaves, Alan J. Hoffman, Uriel G. Rothblum, and Hans Schneider. Line-sum-symmetric scalings of square nonnegative matrices. *Mathematical Programming Studies*, 25:124–141, 1985.

[2] Jack Edmonds and D.R. Fulkerson. Bottleneck extrema. *Journal of Combinatorial Theory*, 8:299–306, 1970.

[3] Gernot M. Engel and Hans Schneider. Diagonal similarity and equivalence for matrices over groups with 0. *Czechoslovak Mathematical Journal*, 25:389–403, 1975.

[4] Mark Hartmann and Michael H. Schneider. *An analogue of Hoffman's circulation theorem for max-balanced flows.* OR Group Report Series #90-2, The Johns Hopkins University, Baltimore, Maryland 21218, 1989.

[5] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[6] R. Tyrrell Rockafellar. *Network Flows and Monotropic Optimization.* John Wiley & Sons, Inc., 1984.

[7] Uriel G. Rothblum, Hans Schneider, and Michael H. Schneider. *Characterizations of Max-Balanced Flows.* OR Group Report Series #90–1, The Johns Hopkins University, Baltimore, Maryland 21218, 1989.

[8] Hans Schneider and Michael H. Schneider. *Max-balancing weighted directed graphs and matrix scaling.* OR Group Report Series #89-4, The Johns Hopkins University, Baltimore, Maryland, 21218, 1989. To appear in *Mathematics of Operations Research.*

[9] Hans Schneider and Michael H. Schneider. Towers and cycle covers for max-balanced graphs. *Congressus Numerantium*, 73:159–170, January 1990.

[10] Michael H. Schneider. Matrix scaling, entropy minimization, and conjugate duality (I): Existence conditions. *Linear Algebra and Its Applications*, 114/115:785–813, 1989.

[11] Michael H. Schneider. Matrix scaling, entropy minimization, and conjugate duality (II): The dual problem. *Mathematical Programming, Series B*, 46, 1990.

HILBERT BASES, CARATHEODORY'S THEOREM AND COMBINATORIAL OPTIMIZATION

András Sebő
IMAG, ARTEMIS, Université Fourier Grenoble 1, BP 53X
38041 GRENOBLE, Cedex, FRANCE

Abstract : A Hilbert basis is a set of vectors with the property that every integer vector in the cone generated by this set is also a nonnegative integer combination of its elements. Hilbert bases were defined by Giles and Pulleyblank (1979) to study total dual integrality. They come up in a natural way in different combinatorial optimization problems from matroid bases through totally dual integral inequality systems to matchings, arborescences or multicommodity flows, and formulate the pure algebraic essence of certain properties of these.

In this paper we are studying some structural properties of Hilbert bases. The main goal is to prove a Caratheodory type statement, a problem raised by a celebrated work of Cook, Fonlupt and Schrijver (1987). Proving it in some special cases we would like to show a new kind of approach to this problem. These special cases include combinatorial examples for which the "integral Caratheodory theorem" may be of interest for its own sake.

We would also like to show that the effect of this problem is beyond the integral Caratheodory problem: the main conjecture contains other results about totally dual integral systems, and more generally would become a basic structural property of combinatorial objects for which integer minimax theorems hold.

1. Introduction

The origins of the problem we are going to study lie in a seemingly innocent question of Cunningham (1987) related to testing membership in matroid polyhedra: *if a vector can be written as a non-negative integer combination of (characteristic vectors of) matroid bases, can it also be expressed as a non-negative integer combination of a small number of matroid bases.* Cunningham gave a first answer to this question that was satisfactory for his goals: if the ground-set of the matroid has n elements, a polynomial number $O(n^4)$ of matroid bases are always enough.

It is an easy consequence of Edmonds' (1970) matroid partition theorem that matroid bases have the property that any vector that can be written both as their non-negative and their integer combination can also be written as their non-negative integer combination. This property turned out to be the only interesting one from the point of view of Cunningham's above mentioned question (see Cook, Fonlupt, Schrijver (1986)). It is actually the algebraic essence of different combinatorial objects as it was shown by the papers referred to above, and as we would like to point out later.

The set of non-negative (real) combinations of the vectors $a_1, ..., a_k$ is called the

cone generated by these vectors and will be denoted by cone$(a_1, ..., a_k)$. "*cone*" will always mean polyhedral (that is finitely generated) cone. A cone is called *pointed*, if it does not contain any linear subspace besides the 0-space, or equivalently, if there exists a hyperplane such that the only element of the cone on the hyperplane is the 0, or equivalently, if the 0 vector cannot be written as a non-negative combination of the coefficient vectors of the linear inequalities describing the cone.

The lattice generated by the vectors $a_1, ..., a_k \in \mathbb{Z}^n$ is the set of their integer combinations, and will be denoted by lat$(a_1, ..., a_k)$. The *basis* of a lattice is a set of linearly independent vectors which generate the lattice. It is well-known that every lattice has a basis (cf. eg. Schrijver(1986)). det$(a_1, ..., a_n)$ will denote the determinant of the matrix whose columns are $a_1, a_2, \ldots, a_n$.

The *parallelepiped* defined by the integer vectors $a_1, \ldots, a_k \in \mathbb{Z}^n$ will be the following set par$(a_1, \ldots, a_k)$ of integer vectors:

$$\text{par}(a_1, \ldots, a_k) := \{w = \sum_{i=1}^{k} \lambda_i a_i : 0 \leq \lambda_i < 1 \quad (i = 1, \ldots, k), \quad w \text{ integer } \}.$$

Motivated by total dual integral systems, Giles and Pulleyblank (1979) defined Hilbert bases, they and Schrijver (1981) proved some basic properties of them, and showed their relation to total dual integrality. Later works such as Cunningham (1987), Cook, Fonlupt and Schrijver (1986), or Lovász (1987) show that the significance of Hilbert bases is beyond totally dual integral systems. They play an important role in integer programming in general, for example in the Chvátal closing procedure (cf. Schrijver (1986)).

The finite set H will be called *Hilbert-generating-system*, if every vector in cone$(a_1, ..., a_k) \cap$ lat$(a_1, ..., a_k)$ can also be written as a non-negative integer combination of $a_1, ..., a_k$. A *Hilbert basis* is a minimal Hilbert-generating-system of a given cone and lattice, that is H is a Hilbert basis if and only if it is a Hilbert generating system, and the cone generated by any proper subset of it is either not a Hilbert-generating-system or generates a smaller lattice or a smaller cone. The above mentioned property of matroid bases means exactly that they form a Hilbert generating system. Furthermore, since no matroid basis can be a non-negative combination of others, they form a Hilbert basis. (We do not distinguish subsets of a set from their characteristic (incidence) vectors).

Remark: This terminology slightly differs from that used in some other papers, but reflects somewhat the present folklore:
1. The term "Hilbert basis" was used earlier for Hilbert generating systems. Since bases in algebra are minimal generating systems, and since Schrijver (1981) has shown that pointed cones have a unique "minimal Hilbert basis" (see below), it was more and

more used for minimal systems as well, which causes some confusion.
2. In some papers Hilbert bases are restricted to satisfy $\text{lat}(H) = \mathbb{Z}^n$, though not all the examples satisfy this restriction (for example matroid bases or matchings do not). This is not bad: if $\text{lat}(H) \neq \mathbb{Z}^n$, use the linear transformation which brings a basis of $\text{lat}(H)$ into the unit vectors. This linear transformation brings H to a Hilbert basis H' "isomorphic" to the original one, and $\text{lat}(H') = \mathbb{Z}^n$. From now on *we shall also suppose that a Hilbert basis H satisfies* $\text{lat}(H) = \mathbb{Z}^n$, and if we want to emphasize that an example does not satisfy this additional restriction we shall signal it.

Les Trotter (1987) has put the question of finding examples of "general TDI-systems" related to Hilbert bases without the assumption $\text{lat}(H) = \mathbb{Z}^n$ in exactly the same way as TDI systems to Hilbert bases with this assumption (see Section 4). He has also found a nice example of "general TDI systems" which are not TDI in the usual sense, shown in Section 4.

We finish this introduction by three simple results and a series of conjectures about Hilbert bases that will play an important role in the sequel.

The *Hilbert generating system* or Hilbert basis of a cone C is a Hilbert generating system, or Hilbert basis of H with $C = \text{cone}(H)$ and $\text{lat}(H) = \mathbb{Z}^n$. In other words a Hilbert generating system of C is a finite set $H \subseteq C$ with the property that every integer vector in C is can be expressed as a non-negative integer combination of H; a Hilbert basis of C is an (inclusionwise) minimal Hilbert generating system of C. (An arbitrary Hilbert basis H (with $\text{lat}(H) = \mathbb{Z}^n$) is the Hilbert basis of $\text{cone}(H)$.) The following result is due to Giles and Pulleyblank (1979) :

Theorem 1.1 *Every cone has a finite Hilbert generating system.*

Proof. Let $C = \text{cone}(a_1, \ldots, a_k)$. $\text{par}(a_1, \ldots, a_k)$ is clearly a finite set, because it is bounded and it contains only integer vectors. But $\{a_1, \ldots, a_k\} \cup \text{par}(a_1, \ldots, a_k)$ is a Hilbert generating system of the cone C, because if $w \in C$, $w = \sum_{i=1}^{k} \lambda_i a_i$, then

$$w - \sum_{i=1}^{k} \lfloor \lambda_i \rfloor a_i = \sum_{i=1}^{k} \{\lambda_i\} a_i \in \text{par}(a_1, \ldots, a_k)$$

where $\lfloor x \rfloor$ denotes the integer part of the number x, and $\{x\}$ denotes its fractional part. Q.E.D.

Schrijver (1981) proved the following theorem:

Theorem 1.2 *Every pointed cone has a unique Hilbert basis.*

The existence of a Hilbert basis follows from the previous theorem. The following proof of the unicity was pointed out to me by Brahim Chaourar. (I learnt from Les Trotter that Jiyong Liu proved it in a similar way.)

Proof. Suppose the rows of the $k \times n$ matrice P and the $l \times n$ matrix Q both form Hilbert-bases of one and the same pointed cone C: $P = AQ$ and $Q = BP$, where A and B are non-negative integer matrices of size $k \times l$ and $l \times k$ respectively. We can deduce $P = ABP$.

We first show that AB is the identity matrix. Denoting the j-th element of its i-th row by λ_{ij}, our matrix equation is equivalent to the equations $p_i = \sum_{j=1}^{k} \lambda_{ij} p_j$, where p_i is the i-th row of P and $\lambda_{ij} \geq 0$ is integer ($i = 1, \ldots, k$, $j = 1, \ldots, k$). For every $i = 1, \ldots, k$, $\lambda_{ii} = 1$, for say $\lambda_{11} = 0$ would contradict the fact that the p_i-s form a *minimal* Hilbert generating system (we can delete p_1); $\lambda_{11} > 1$ would contradict the *pointedness* of C. Substituting this into our equation we get $0 = \sum_{j \neq i} \lambda_{ij} p_j$ for every i, and since C is pointed, and the coefficients are non-negative, we can deduce $\lambda_{ij} = 0$ if $i \neq j$.

We conclude that A, B are non-negative integer matrices, and AB is the $k \times k$ identity matrix. Since the rows of P and Q form a *minimal* Hilbert generating system, A and B have no zero columns and rows. It follows immediately that both A and B are permutation matrices.

Q.E.D.

From now on *we shall suppose that Hilbert generating systems and Hilbert bases generate pointed cones,* except if we emphasize the countrary. A cone will also automatically mean *pointed cone.* All the examples we shall mention treat only pointed cones as well.

Theorem 1.2 immediately implies the following observation of Schrijver (1981). (This played an important role in his original proof.)

Corollary *The Hilbert basis of the cone* $C = \mathrm{cone}(a_1, \ldots, a_k)$ *is the set*

$$H = \{h \in C \cap \mathbb{Z}^n \setminus \{0\} : h \text{ is not the sum of two non-zero integer vectors of } C\}.$$

Clearly, $H \subseteq \mathrm{par}(a_1, \ldots, a_k) \cup \{a_1, \ldots, a_k\}$.

Thus the formula in the above corollary is an equivalent definition of the *Hilbert basis of a cone* C, and it will be used as such, without any more reference to it.

Cook, Fonlupt and Schrijver (1986) have proved the following Caratheodory type theorem for Hilbert bases *:

Theorem 1.3 *Let* C *be a pointed cone, and let* $H \subseteq \mathbb{Z}^n$ *be its Hilbert basis. If* $w \in \mathrm{cone(H)} \cap \mathbb{Z}^n$ *then* w *is the positive integer linear combination of at most* $2n - 1$

* Recall Caratheodory's theorem for cones: *every element of a cone can be written as the non-negative linear combination of at most* n *generating vectors of the cone.*, see Schrijver (1986) Corollary 7.1i.

elements of H. If H consists only of 0-1 *vectors, then this bound can be improved to* $2n-2$.

We have to put the proof here because the proof of Theorem 2.1 will refer to its details.

Proof. Let $H = \{a_1, \ldots, a_k\}$, $w = \sum_{i=1}^{k} \lambda_i a_i$, and suppose $\sum_{i=1}^{k} \lambda_i$ is maximum among all possible choices. (This maximum is finite because C is pointed.) We know from linear programming that the set $\{a_i : \lambda_i > 0\}$ can be chosen to be linearly independent (a version of Caratheodory's theorem)*. We can thus suppose without loss of generality that $\lambda_i = 0$ if $i > n$, that is $w = \sum_{i=1}^{n} \lambda_i a_i$.

Let

$$w_0 = w - \sum_{i=1}^{n} \lfloor \lambda_i \rfloor a_i = \sum_{i=1}^{n} \{\lambda_i\} a_i \in \mathrm{par}(a_1, \ldots, a_n) \tag{1.1}$$

$w_0 \in C$, and w_0 is an integer vector, whence there exist $\alpha_i \geq 0$ integers such that $w_0 = \sum_{i=1}^{k} \alpha_i a_i$. Clearly,

$$w = \sum_{i=1}^{k} \alpha_i a_i + \sum_{i=1}^{n} \lfloor \lambda_i \rfloor a_i \tag{1.2}$$

$\sum_{i=1}^{k} \alpha_i \leq n-1$, for if not, $\sum_{i=1}^{k} \alpha_i \geq n > \sum_{i=1}^{k} \{\lambda_i\}$, and thus the sum of the coefficients in (1.2) is greater than $\sum_{i=1}^{n} \lambda_i$, a contradiction.

Since the α_i-s are integers $|\{i : \alpha_i > 0\}| \leq n-1$ follows, proving that the number of positive coefficients in (1.2) is at most $2n-1$.

Q.E.D.

Cook Fonlupt and Schrijver (1986) add the remark that in 2 dimensions 2 elements are enough, and they do not have any example where n elements would not be enough in general instead of the $2n-1$ above.

Conjecture A *Let $H \subseteq \mathbb{Z}^n$ be a Hilbert basis, and $w \in \mathrm{cone(H)} \cap \mathbb{Z}^n$. Then w is the positive integer linear combination of at most n elements of H, and these can be determined in polynomial time.*

In a lecture, Bill Cook communicated an additional fact which was very important from the point of view of the present work: in the plane, the determinant of neighbouring Hilbert basis elements is ± 1, (equivalently, they generate the lattice of all integers, or just the same lattice as H), for a proof see the end of this introduction. In sections

---

* In the language of linear programming: there exists an optimal *basic* solution.

2 and 3 we are going to prove generalizations of this fact, in Sections 4 and 5 we shall deduce some consequences for combinatorial problems. These results, many examples and the strong belief in the beauty of nature makes us think that the same is true in general. In other words, if the following conjecture is true, it should be due to Cook Fonlupt and Schrijver (1986), if it is not, the responsibility is the author's.

CONJECTURE B *If $H \subseteq \mathbb{Z}^n$ is a full dimensional pointed Hilbert basis, then* $\text{cone}(H)$ *is covered by cones* $\text{cone}(a_1, \ldots, a_n)$, *where* $\{a_1, \ldots, a_n\} \subseteq H$, *and* $\det(a_1, ..., a_n) = \pm 1$; $a_1, \ldots, a_n$ *such that* $\text{cone}(a_1, \ldots, a_n)$ *contains a given element of* $\text{cone}(H) \cap \mathbb{Z}^n$, *and* $\det(a_1, ..., a_n) = \pm 1$ *can be computed in polynomial time.*

It is easy to see that Conjecture B implies Conjecture A. For general Hilbert-bases, instead of "$\det(a_1, ..., a_n) = \pm 1$" we have to write simply that "$(a_1, ..., a_n)$ is a basis of $\text{lat}(a_1, ..., a_n)$."

Note that vectors $a_1, \ldots, a_n$ with $\det(a_1, ..., a_n) = 1$ are the minimum cardinality full dimensional Hilbert bases and they are the only linearly independent Hilbert bases. For a not necessarily full dimensional set of linearly independent vectors $\{a_1, \ldots, a_k\}$ the equivalence of the following statements can be shown easily:

(i) $\{a_1, \ldots, a_k\}$ *is linearly independent and is a Hilbert basis.*

(ii) $\text{par}(a_1, ..., a_k) = \{0\}$

(iii) The g.c.d. of the $k \times k$ *subdeterminants of the matrix whose columns are* $a_1, \ldots, a_k$ *is 1.* (The equivalence of (ii) with the rest relies however on some knowledge on lattices, see Schrijver (1986). Actually, (iii) will be used only for $k = n$, when it is evident.)

For simplicity, we shall often suppose that our Hilbert basis is of full rank. This is not a restriction of the generality: if the Hilbert basis is in a subspace of rank r, choose a basis of this subspace, and represent the vectors of the subspace in with the coefficient vectors of linear combinations of this basis. This gives a full dimensional representation of the same Hilbert basis in $\mathbb{R}^r$.

This argument permits to extend statements from the full dimensional case to arbitrary Hilbert bases. For example, using that full dimensional linearly independent Hilbert bases are exactly the sets of vectors with determinant ± 1, we get the following obviously equivalent version of Conjecture B, which does not need the assumption about the full rank, and does not speak about determinants:

CONJECTURE C *Let $H \subseteq \mathbb{Z}^n$ be a Hilbert-basis where* $\text{cone}(H)$ *is pointed. If H is linearly dependent, and $w \in \text{cone}(H) \cap \mathbb{Z}^n$, then there exists a Hilbert-basis $H' \subset H$, ($H' \neq H$), $w \in \text{cone}(H')$, and H' can be computed in polynomial time.*

The reader may find it useful to study numerical examples of Hilbert bases first in 2 dimensions. Taking two linearly independent relatively prime vectors with non-± 1 determinant, clearly, not every integer vector of the pointed cone C generated by these two vectors is an integer combination of the generating vectors. You can easily find the (uniquely determined) elements of the Hilbert basis of C. The experience aquired through such examples will probably help to follow the proof below, and arguments all along the paper.

Let us finish this introduction by proving these conjectures for $n = 2$. This case is easy, but it isn't a completely banal exercise.

However, short proofs can be given in various ways. Several proofs can be extracted from more general arguments, for example Lemma 1 of Theorem 2.2 below gives a proof. The particularity of this case, exploited by the proofs is that *neighboring Hilbert basis elements have determinant* 1, *or equivalently, if we delete either of the extreme rays, the remaining vectors form a Hilbert basis again.* It is not difficult to prove that the Hilbert basis H of the cone generated by the integer vectors a_1, a_2 lies in $\mathrm{conv}(a_1, a_2, 0)$. (This fact implies immediatly Conjecture C: $H \setminus \{a_1\}$ is a Hilbert basis, for if $v \in \mathrm{conv}(H \setminus \{a_1\})$, and $v = v_1 + v_2$, $v_1, v_2 \in \mathrm{cone}(a_1, a_2)$, then clearly, $v_1, v_2 \in \mathrm{conv}(H \setminus \{a_1\})$.) Our purpose with giving a separate proof here is that some aspects of Hilbert bases might be easier to understand on this simple example. The following version, simplified down to the bare essentials, was exhibited by Péter E. Soltész:

Proof of Conjecture C for $n = 2$. If $|H| \leq 2$, the statement is trivial. Let $a_1, a_2 \in H$ be the extreme rays of $\mathrm{cone}(H)$. By our assumptions a_1 and a_2 are linearly independent and $H \setminus \{a_1, a_2\} \neq \emptyset$. It is enough to prove that $H \setminus \{a_1\}$ is a Hilbert basis, for then by symmetry $H \setminus \{a_2\}$ is also a Hilbert basis, and arbitrary $w \in \mathrm{cone}(H)$ is contained in the cone generated by one of these.

One of the extreme rays of $\mathrm{cone}(H \setminus \{a_1\})$ is a_2, let the other be a_1'. Of course $a_1' \in H$. All we have to prove is that every $w \in \mathrm{par}(a_1', a_2)$ is the non-negative integer combination of vectors in $H \setminus \{a_1\}$. We shall actually prove even more. Let $w \in \mathrm{par}(a_1', a_2)$. We know that w can be written as the non-negative integer combination of vectors in H: $w = \sum_{h \in H} \alpha(h) h$, where $\alpha(h) \geq 0$ integer for all $h \in H$. We shall prove that in *every* combination of this form $\alpha(a_1) = 0$. Suppose indirectly that $\alpha(a_1) \geq 1$, and substitute every $h \in H$ by their expression as a nonnegative linear combination of a_1 and a_2: we get that

$$w = \beta_1 a_1 + \beta_2 a_2, \text{ where } \beta_1 \geq 1, \beta_2 \geq 0. \tag{1.3}$$

On the other hand $w = \lambda_1' a_1' + \lambda_2' a_2$ $(0 \leq \lambda_1', \lambda_2' < 1)$, where $a_1' = \lambda_1 a_1 + \lambda_2 a_2$

$(0 \leq \lambda_1, \lambda_2 < 1)$, whence

$$w = \lambda_1' \lambda_1 a_1 + (\lambda_1' \lambda_2 + \lambda_2') a_2. \tag{1.4}$$

In (1.3) the coefficient of a_1 is at least 1, whereas in (1.4) it is smaller than 1. This is a contradiction, because there is a unique way to express w in the basis a_1, a_2.

Q.E.D.

The reader could visualize this proof finding the clear geometric meaning of each step.

2. Improving by 1

In this section we improve only by 1 three known bounds. The proofs seem to require new methods though, and may give more general indications for attacking the conjectures A, B, C. Their proofs are strictly related to Conjectures B and C. The main support for the conjectures is just the following fact valid for the whole paper: *in all cases when Conjecture A is true, the stronger Conjectures B and C also hold; the same proofs work for them; furthermore, in most cases, the only way of proving Conjecture A is to prove Conjectures B or C.*

Theorem 2.1 *Let $H \subseteq \mathbb{Z}^n$ be the Hilbert basis of a pointed cone, and $w \in \text{cone}(H) \cap \mathbb{Z}^n$. Then w is the positive integer linear combination of at most $2n-2$ elements of H.*

The proof, in addition to the proof of Theorem 1.3, exploits the symmetry of parallelepipeds:

Proof. Let $H = \{a_1, \ldots, a_k\}$, $w = \sum_{i=1}^k \lambda_i a_i$; $\sum_{i=1}^k \lambda_i$ is maximum; $a_1, \ldots, a_n$ are linearly independent, $\{a_i : \lambda_i > 0\} = \{a_1, \ldots, a_n\}$. (If $|\{i : \lambda_i > 0\}| < n$, then the statement follows already from the proof of Theorem 1.3.) *Assume in addition that* $\text{conv}(a_1, \ldots, a_n) \cap H = \{a_1, \ldots, a_n\}$. We can assume this without loss of generality, for

$$w = \varepsilon h + \sum_{i=1}^n (\lambda_i - \varepsilon \gamma_i) a_i, \tag{2.1}$$

and if $h = \sum_{i=1}^n \gamma_i a_i$ with $\sum_{i=1}^n \gamma_i = 1$, $(0 \leq \gamma_i < 1, i = 1, \ldots, n)$, then the sum of the coefficients in such a combination remains equal to $\sum_{i=1}^n \lambda_i$. The choice $\varepsilon := \min\{\frac{\lambda_i}{\gamma_i} : i = 1, \ldots, n\} =: \frac{\lambda_j}{\gamma_j}$ makes clear that $\text{cone}(a_1, \ldots, a_{j-1}, h, a_{j+1}, \ldots, a_n)$ contains w, and combines it with the same some of coefficients*.

* This is just a "pivot" bringing h into the "basis", without changing the "objective value", because the "relative cost" of h was 0.

Let w_0 be defined by (1.1). We have now

$$\sum_{i=1}^{n}\{\lambda_i\} < n-1. \tag{2.2}$$

Indeed, suppose indirectly that $\sum_{i=1}^{n}\{\lambda_i\} \geq n-1$, *that is, where* $\gamma_i := 1-\{\lambda_i\} > 0$, $(i = 1,\ldots,n)$

$$\sum_{i=1}^{n}\gamma_i \leq 1. \tag{2.3}$$

Let now

$$h := \sum_{i=1}^{n}\gamma_i a_i = a_1 + \ldots + a_n - w_0 \in \mathrm{cone}(a_1,\ldots,a_n)\cap\mathbb{Z}^n,$$

furthermore, if $h \notin H$ substitute h in (2.1) by a non-negative integer combination of Hilbert basis elements. We must have equality in (2.3), because otherwise the sum of the coefficients in (2.1) is bigger than $\sum_{i=1}^{n}\lambda_i$ for any positive ε; $h \in H$ for the same reason. Thus $h \in \mathrm{conv}(a_1,\ldots,a_n)\setminus\{a_1,\ldots,a_n\}$, contradicting the assumption made in the beginning of the proof, and proving (2.2).

After these remarks we proceed in exactly the same way as in the proof of Theorem 1.3. However, because of (2.2), we have now the following tighter bound in the last paragraph of the proof of Theorem 1.3: $\sum_{i=1}^{k}\alpha_i \leq n-2$, for if not, $\sum_{i=1}^{k}\alpha_i \geq n-1 > \sum_{i=1}^{n}\{\lambda_i\}$, and thus the sum of the coefficients in (1.2) is greater than $\sum_{i=1}^{n}\lambda_i$, a contradiction.

Since the α_i-s are integers $|\{i : \alpha_i > 0\}| \leq n-2$ follows, proving that the number of positive coefficients in (1.2) is at most $2n-2$.

Q.E.D.

Remark:

1.Instead of requiring $\mathrm{conv}(a_1,\ldots,a_n)\cap H = \{a_1,\ldots,a_n\}$ in the proof, we could have assumed that $\det(a_1,\ldots,a_n)$ is minimal:

$$\det(a_1,\ldots,a_{j-1},h,a_{j+1},\ldots,a_n) = \det(a_1,\ldots,a_{j-1},\sum_{i=1}^{n}\gamma_i a_i,a_{j+1},\ldots,a_n) = \gamma_j\det(a_1,\ldots,a_n)$$

Thus "pivoting" decreases the determinant, if we "bring" an element of the parallelepiped into the basis.

This trick, although it is more technical than the one we used in the above version of the proof, can also be used to prove other statements. For instance *it implies immediately the existence of a Hilbert generating system for which Conjecture B holds*:

by Caratheodory's theorem cone(H) is covered by cones of the type cone(H') (where $H' \subseteq H$ is linearly independent). If for each of these H' we take $h \in \text{par}(H')$ and replace H' by the n cones that have h and $n-1$-elements of H' as extreme rays (we suppose H is full dimensional), we get a new covering by cones, and by the above remark the maximum determinant has decreased. Repeating this a finite number of times, we get the appropriate Hilbert generating system.

2. A third possibility, which is *better from the algorithmic point of view*: let $c_i := 1 - \varepsilon \parallel a_i \parallel$, where ε is "sufficiently small"; it is easy to see that a basis with maximal objective value with respect to this objective function satisfies $\text{conv}(a_1, \ldots, a_n) \cap H = \{a_1, \ldots, a_n\}$. It follows that the positive linear combination with at most $2n-2$ Hilbert basis elements can be determined in polynomial time.

The following theorem will of course not contain many combinatorial examples. (It could be interesting in itself though from the viewpoint of three dimensional geometry.) However, its proof is probably the one which goes the deepest into the structure of Hilbert bases in general:

Theorem 2.2 *Conjectures A, B, and C are true for $n \leq 3$.*

To prepare the proof we state two lemmata valid in arbitrary dimension:

Lemma 1 *Let $H \subseteq \mathbb{Z}^n$ be the Hilbert basis of a pointed cone, $H = a_1, \ldots, a_k$ and $w \in \text{cone(H)} \cap \mathbb{Z}^n$. Then there exist coefficients λ_i such that $w = \sum_{i=1}^k \lambda_i a_i$, and*

(i) $\sum_{i=1}^k \lambda_i$ *is maximum under this constraint.*

(ii) $\{a_i : \lambda_i > 0\}$ *is linearly independent, say* $\{a_i : \lambda_i > 0\} = \{a_1, \ldots, a_s\}$, $s \leq n$.

(iii) If $h \in \text{par}(\{a_1, \ldots, a_s\} \setminus \{0\})$, $h = \sum_{i=1}^k \gamma_i a_i$, *then*

$$1 < \sum_{i=1}^{k} \gamma_i < s - 1 \leq n - 1. \tag{2.4}$$

Less formally, there exists an "optimal basis" for which (iii) holds. Note that Conjecture B for $n = 2$ follows immediately.

Proof. Let $H = \{a_1, \ldots, a_k\}$, $w = \sum_{i=1}^k \lambda_i a_i$; $\sum_{i=1}^k \lambda_i$ is maximum; $\{a_i : \lambda_i > 0\} = \{a_1, \ldots, a_s\}$, and this set is linearly independent (see the proof of Theorem 1.3. *Assume in addition that* $\text{conv}(a_1, \ldots, a_s) \cap H = \{a_1, \ldots, a_s\}$ like in the proof of Theorem 2.1.

Thus (i) and (ii) hold by the above choice. Let $h \in \text{par}(a_1, \ldots, a_s)$, $h = \sum_{i=1}^s \gamma_i a_i$. We have to prove (2.4). Like in the proof of Theorem 2.1, (2.3) cannot hold, whence $\sum_{i=1}^s \gamma_i > 1$. Applying the same to the symmetric $\bar{h} = a_1 + \ldots + a_s - h = \sum_{i=1}^s (1-\gamma_i) a_i$ we get that $\sum_{i=1}^s 1 - \gamma_i > 1$ and (2.4) is proved.

Q.E.D.

The following Lemma is well-known from the geometry of numbers. For the sake of completeness we sketch a proof using the Hermite normal form:

Lemma 2 *Let $a_1, \ldots, a_n \in \mathbb{Z}^n$ be a basis of $\mathbb{R}^n$. Then* $|\mathrm{par}(a_1, \ldots, a_n)| = |\det(a_1, \ldots, a_n)|$ (For a general set $a_1, \ldots, a_k$ we simply replace the determinant by the greatest common divisor of the $r \times r$ determinants where r is their rank.)

Proof. Let A be the matrix whose columns are $a_1, \ldots, a_n$. Let r_i denote the i-th row of A. Clearly, if we replace a row r_i by $r_i \pm r_j$ $(i \neq j)$, then the set of vectors $\underline{\lambda} = (\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^n$ for which $A\underline{\lambda}$ is integer remains the same. In particular, the number of elements in the parallelepiped defined by the new column vectors remains the same; the determinant does not change either. With such operations one can arrive to the "Hermite normal form" (see Schrijver (1986) p.45) of the rows. We can thus suppose that A is lower triangular. Let the elements in its main diagonal be $d_1, \ldots d_n$. Clearly, for $A\underline{\lambda}$ to be in $\mathrm{par}(a_1, \ldots, a_n)$ we have d_1 different choices for λ_1: $\frac{t}{d_1}$ $(t = 0, \ldots, d_1 - 1)$. Similarly, if $\lambda_1, \ldots, \lambda_{i-1}$ have already been chosen, and the i-th component of $\sum_{j=1}^{i-1} \lambda_j a_j$ is x, then the possible choices for λ_i are $\frac{\lceil x \rceil - x + t}{d_i}$ $(t = 0, \ldots, d_i - 1)$: for all possible choices of $\lambda_1, \ldots, \lambda_{i-1}$ we have d_i choices for λ_i. We conclude that $\mathrm{par}(a_1, \ldots, a_n)$ has $d_1 \ldots d_n = det(A)$ elements.

Q.E.D.

Proof of Theorem 2.2. Suppose $n = 3$, and let us prove Conjecture C.

Claim 1 *If $H \subseteq \mathbb{Z}^3$ is a Hilbert-basis, and $w \in \mathrm{cone(H)} \cap \mathbb{Z}^3$, then there exist vectors $a_1, a_2, a_3 \in H$ such that $\mathrm{par}(a_1, a_2, a_3) \setminus \{0\} \subseteq \mathrm{H}$, and $w \in \mathrm{cone}(a_1, a_2, a_3)$.*

Indeed, let $\{a_1, a_2, a_3\}$ and $\{\lambda_1, \lambda_2, \lambda_3\}$ be the basis and coefficients provided by Lemma 1 (s=3), and $h \in \mathrm{par}(a_1, a_2, a_3)$, $h = \gamma_1 a_1 + \gamma_2 a_2 + \gamma_3 a_3$. By Lemma 1 (2.4), $\gamma_1 + \gamma_2 + \gamma_3 < 2$. If $h \notin H$, then substituting h with a non-negative integer combination of H, the sum of the coefficients in (2.1) is again bigger than $\lambda_1 + \lambda_2 + \lambda_3$ contradicting Lemma 1 (i).

Claim 2 *If Claim 1 does not hold for any proper subset of H (it must hold then for H, that is $\mathrm{par}(a_1, a_2, a_3) \setminus \{0\} = H$), then it has an element $h = \gamma_1 a_1 + \gamma_2 a_2 + \gamma_3 a_3$ that satisfies the equation*

$$\Delta(\gamma_1 + \gamma_2 + \gamma_3) = \Delta + 1,$$

where $\Delta := \det(a_1, a_2, a_3)$.

Before proving Claim 2, let us summarize our knowledge about the parallelepiped $\mathrm{par}(a_1, a_2, a_3)$: it has $\Delta - 1$ different non-zero elements (see Lemma 2) of the form

$h = \gamma_1 a_1 + \gamma_2 a_2 + \gamma_3 a_3$, where the coefficients are rational numbers all with Δ as denominator (Cramer's rule); denoting the sum of the numerators of the coefficients by $s(h)$,

$$s(h) = \Delta(\gamma_1 + \gamma_2 + \gamma_3). \tag{2.5}$$

On the other hand, (2.4) for $n = 3$ gives $\Delta < s(h) < 2\Delta$, in other words,

(2.6) *$s(h)$ is always one of the values $\Delta + 1, \ldots, 2\Delta - 1$, that is one of $\Delta - 1$ different values.*

We shall prove that

(2.7) *for $h_1 \neq h_2 \in \mathrm{par}(a_1, a_2, a_3)$, $s(h_1) \neq s(h_2)$.*

It follows then by (2.6) that for each $i \in \{\Delta + 1, \ldots, 2\Delta - 1\}$, $s(h) = i$ is satisfied for exactly one $h \in H$, and in particular $s(h) = \Delta + 1$ for some $h \in H$. Because of (2.5) this is just the statement of Claim 2.

Let $h_1, h_2 \in \mathrm{par}(a_1, a_2, a_3)$, $s(h_1) = s(h_2) =: s$. All we have to prove, is $h_1 = h_2$.

For $h \in \mathrm{par}(a_1, a_2, a_3)$ let $\bar{h} := a_1 + a_2 + a_3 - h$. Clearly, $\bar{h} \in \mathrm{par}(a_1, a_2, a_3)$. The identity

$$h_1 + \bar{h}_2 = a_1 + a_2 + a_3$$

is obviously equivalent to $h_1 = h_2$. $s(h_1) + s(\bar{h}_2) = s + 3\Delta - s = 3\Delta$. Thus, in the combination $h_1 + \bar{h}_2 = \delta_1 a_1 + \delta_2 a_2 + \delta_3 a_3$, $\delta_1 + \delta_2 + \delta_3 = \frac{3\Delta}{\Delta} = 3$, and

$$0 < \delta_i < 2, \quad (i = 1, 2, 3). \tag{2.8}$$

Since by (2.6) for every vector in the parallelepiped, the sum of the coefficients is non-integer, the fact that $\delta_1 + \delta_2 + \delta_3$ is integer implies that δ_1, δ_2 and δ_3 are all integers, and using (2.8) we have $\delta_1 = \delta_2 = \delta_3 = 1$. Claim 2 is proved.

We show now how Conjecture C follows from the claims. If H is not of the form $\mathrm{par}(\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3) \setminus \{0\}$ then we are done by Claim 1. Thus we can suppose $H = \mathrm{par}(\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3) \setminus \{0\}$, and that the condition of Claim 2 holds. Clearly, for every $h \in \mathrm{par}(a_1, a_2, a_3)$:

$$\mathrm{par}(a_1, a_2, h) \cup \mathrm{par}(a_1, h, a_3) \cup \mathrm{par}(h, a_2, a_3) \supseteq H \setminus \{h\},$$

and at least the origin is contained in the intersection of the three parallelepipeds on the left hand side, whence:

$$\det(\mathrm{a}_1, \mathrm{a}_2, \mathrm{h}) + \det(\mathrm{a}_1, \mathrm{h}, \mathrm{a}_3) + \det(\mathrm{h}, \mathrm{a}_2, \mathrm{a}_3) - 2 \geq \det(\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3) - 1.$$

If h is now the element guaranteed by Claim 2, we have equality here (the members of the sum at the left hand side are the same as those in Claim 2, through elementary operations on determinants). Consequently we have equality in the above set-containment as well. It follows that $\mathrm{par}(a_1, a_2, h) \setminus \{0\}$, $\mathrm{par}(a_1, h, a_3) \setminus \{0\}$ and $\mathrm{par}(h, a_2, a_3) \setminus \{0\}$ partition H, and are Hilbert bases. The theorem is proved. (The statement about the complexity is clear: the solution of a linear programming problem allows to decrease the size of the problem by 2.)

Q.E.D.

Conjecture A in the following special case has been proved earlier by A. Gerards. The following proof is based on arguments similar to the other two proofs of this section, some elements of which will be useful later:

Theorem 2.3 *Conjectures A, B and C are true for Hilbert bases whose linear rank is one less than their cardinality.*

Proof. We shall prove Conjecture B for the set of vectors $H = \{a_1, \ldots, a_{n+1}\}$ of full rank, forming a Hilbert basis. Clearly, up to a scalar multiple, there exists one unique non-zero linear relation

$$\sum_{i=1}^{n+1} \alpha_i a_i = 0. \tag{2.9}$$

For later convenience we state the following Lemma in a somewhat more general form that we need it here:

Lemma : *If $H = \{a_1, \ldots, a_k\}$ is an arbitrary Hilbert-basis, and the linearly independent vectors $a_1, \ldots, a_s \in H$ do not form a Hilbert basis, then*
a. *There exists an equation of the form*

$$\sum_{i \in I} a_i = \sum_{i \notin I} \alpha_i a_i \tag{2.10}$$

where $I \subseteq \{1, \ldots, s\}$, and $\alpha_i \geq 0$ integers ($i \notin I$).
b. *There exists an equation of the form $\sum_{i \in I} \alpha_i a_i = \sum_{i \notin I} \alpha_i a_i$ where $I \subseteq \{1, \ldots, s\}$, $\alpha_i \geq 0$ integers ($i = 1, \ldots, k$), and $\max\{\alpha_i : i = 1, \ldots, s\} < \max\{\alpha_i : i = s+1, \ldots, k\}$.*

Indeed, $a_1 + \ldots + a_s = h + \bar{h}$, where $0 \neq h \in \mathrm{par}(a_1, \ldots, a_s)$. Since both h and $\bar{h}$ are in $\mathrm{cone}(H)$, they can be expressed as a non-negative integer combination of H. Substitute these expressions for h and $\bar{h}$ into the above equation. In the linear expression we have for h we are sure to have an element different from $a_1, \ldots, a_s$. After eventual simplifications because of elements that occur on both sides we get (2.10) and a. is proved.

To prove Lemma b. take simply $h \in \text{par}(a_1, \ldots, a_s)$. Clearly, $h = \frac{1}{q}(p_1 a_1 + \ldots + p_s a_s)$, $0 \leq p_i < q$ $(i = 1, \ldots, s)$. Expressing h as a non-negative integer combination of H, we get that $p_1 a_1 + \ldots + p_s a_s = q \sum_{i=1}^{k} r_i a_i$, where r_i is non-negative integer, and there exists $j > s$ with $r_j \geq 1$. Thus, after simplifying because of elements that occur on both sides we get an equation with all coefficients smaller than q on the left hand side; a_j is on the right hand side, and its coefficient is at least q . The Lemma is proved.

The Lemma tells us a lot about the coefficients in (2.9). Suppose $a_1, \ldots, a_s$ be independent, but not a Hilbert basis. (Otherwise we are done.) By Lemma a. (2.10) holds, and must coincide with (2.9) up to a scalar multiple. Because of the uniqueness of (2.9), Lemma b. applies to (2.9) as well: (2.9) can be supposed to be identical to (2.10) with the additional property, that there exists j with $\alpha_j \geq 2$. Furthermore,

(2.11) *if* $\{a_j : j \in J\}$, $J \subseteq \{1, \ldots, n+1\}$ *is linearly independent and not a Hilbert basis, then* $J \supseteq I$.

(Because J satisfies the same assumption as $\{1, \ldots, s\}$ and consequently (2.10) holds if we replace I by $I' \subseteq J$. By the uniqueness of (2.10), using $\alpha_j \geq 2$ as well, we get $I' = I$.)

Let now $w \in \text{cone}(H)$ be arbitrary. By Caratheodory's theorem, $w = \sum_{i=1}^{s} \lambda_i a_i$, $\lambda_i \geq 0$, and we can suppose that $a_1, \ldots, a_s$ are those in the Lemma. Let $\lambda := \min\{\lambda_i : i \in I\}$. By (2.10) we have

$$w = \sum_{i \in I} (\lambda_i - \lambda) a_i + \sum_{i \notin I} (\lambda_i + \lambda \alpha_i) a_i \tag{2.12}$$

(define $\lambda_i = 0$ for $i > s$). Since $\lambda_i - \lambda = 0$ for some $i \in I$, in (2.12) the indices of the positive coefficients do not contain I. Thus $\{a_i : a_i$ has positive coefficient in (2.12)$\}$ is linearly independent (because (2.10) is the unique linear relation), and according to (2.11), it is a Hilbert basis.

Q.E.D.

A proof of this theorem also follows from our study of "uncrossable Hilbert bases" (Section 3), and another one from the structure of 1-dimensional Hilbert kernels (see a remark in Section 5 after Conjecture E).

3. Strong and uncrossable Hilbert-bases

An equivalent definition of Hilbert generating systems could be the following: H is a Hilbert generating system if and only if for every $w \in \text{cone(H)} \cap \text{lat(H)}$ there exists

a $h \in H$ such that $w - h \in \text{cone(H)}$.

Chandrasekaran and Tamir (1984) studied a property which can be reformulated as follows: for every $w \in \text{cone(H)} \cap \text{lat(H)}$ and $h \in H$, $w - \epsilon h \in \text{cone(H)}$ with $\epsilon > 0$ implies $w - h \in \text{cone(H)}$. The origin of this property lies in Fulkerson's proof (1972) of the Pluperfect Graph Theorem. The same property permitted to Cook, Fonlupt, Schrijver (1986) to prove Conjecture A for the cliques lying on a face of the clique polytope of a perfect graph.

We first suggest a property which will permit to include further combinatorial examples, and is, on the other hand special enough: we will prove Conjectures A, B, C if this property holds. H will be called a *strong Hilbert-basis*, if for every face F of $C := \text{cone(H)}$ there exists an element $h_F \in F$ with the following property: if $w \in C \cap \mathbb{Z}^n$, and F is the minimal face of C containing w, then $w - h_F \in C$. *

h_F with this property will be called a *strong element* of F. (For basic definitions and statements about the structure of polyhedra cf. Schrijver (1986).) The main result of this section is the following theorem:

Theorem 3.1 *If $H \subseteq \mathbb{Z}^n$ is a strong Hilbert basis, then Conjecture A, B, C hold for it.*

Proof. Let $w \in C \cap \mathbb{Z}^n$, and suppose F_1 is the minimal face of C containing w, and $h_1 := h_{F_1}$. Let $\lambda_1 := \max\{\lambda : w - \lambda h_1 \in C\}$. The dimension of the minimal face F_2 cotaining $w - \lambda_1 h_1$ is less than that of F_1. Apply now the same procedure to $w - \lambda h_1 \in C$, and so on. Clearly, we arrive in this way at a series of faces $F_1, \ldots, F_s$ and linearly independent vectors $h_1, \ldots, h_s$ ($h_i = h_{F_i}$, $i = 1, \ldots, s$), $w \in \text{cone}(h_1, \ldots, h_s)$.

We prove now by induction on $\dim(F_1)$ that $h_1, \ldots, h_s$ is a Hilbert basis. If $\dim(F_1) = 1$ the statement is obvious. If $h_1, \ldots, h_s$ is not a Hilbert basis, there exists a $0 \neq h \in \text{par}(h_1, \ldots, h_s)$, $h = \sum_{i=1}^{s} \gamma_i h_i$ ($0 \leq \gamma_i < 1$). By the induction hypothesis $h_2, \ldots, h_s$ is a Hilbert basis, whence $\gamma_1 \neq 0$. The minimal face containing h is F_1, and because of $\sum_{i=2}^{s} \gamma_i h_i \in F_2$ where $\dim(F_2) < \dim(F_1)$ we see that $\gamma_1 = \max\{\lambda : h - \lambda h_1 \in C\}$. But $0 < \gamma_1 < 1$ contradicts the property of $h_{F_1} = h_1$ in the strong Hilbert basis H.

---

* Using Chandrasekaran and Tamir's terminology, strong Hilbert bases are exactly those Hilbert bases H, whose elements have an order such that the lexicographically maximal linear expression of every $h \in \text{cone}(H)$ is integer. (Easy to see.) Chandrasekaran and Tamir investigated problems where the same holds for *every* order.

The claim about the complexity follows if we choose objective values $c(h_F) := \Delta^{\dim(F)}$, where Δ is the biggest determinant a square submatrix of H can have. *

Q.E.D.

A second property that will imply the conjectures: uncrossability. Recall the following from the Lemma a. of Theorem 2.3: *If H is a Hilbert basis, and the linearly independent vectors $a_1, \ldots, a_s \in H$ do not form a Hilbert-basis, then their sum can also be written in another way as a non-negative integer combination of Hilbert-basis elements.* We shall say that H is *uncrossable*, if for some objective function $c : H \longrightarrow \mathbf{R}$ and for every independent subset $\{a_1, \ldots, a_s\} \subseteq H$, among the combinations

$$\sum_{h \in H} \alpha(h)h = a_1 + \ldots + a_s, \tag{3.1}$$

($\alpha(h)$ integer for every $h \in H$), there is one with

$$\sum_{h \in H} \alpha(h)c(h) > c(a_1) + \ldots + c(a_s). \tag{3.2}$$

(This is a generalization of the properties used in "uncrossing procedures" for many combinatorial problems. Eg. the square of the cardinality of a set is often used as objective function.)

CONJECTURE D *Every Hilbert basis is uncrossable, and the objective function c in the definition of uncrossability is computable in polynomial time.*

The reader can check that *in all the special cases for which we proved the conjectures, Conjecture D also holds.* Check it for instance for the example of Theorem 2.3: any function c such that the objective value of the left hand side of (2.10) is smaller than that of the right hand side will obviously do. Thus the proof of Theorem 3.2 below will provide a new proof for Theorem 2.3 . An other example: the choice of the objective function at the end of the proof of Theorem 3.1 prove that *strong Hilbert bases are uncrossable* (Proofs in this paper can be considered to exhibit an appropriate objective function to show uncrossability, and Theorem 3.2 below proves again the conjectures for every case.) Conjecture D is important from the algorithmic point of view.

Theorem 3.2 *Conjectures A, B, C hold for uncrossable Hilbert bases*

Proof. Let H be an uncrossable Hilbert basis, $c : H \longrightarrow \mathbb{R}$ the function that ensures uncrossability, and $b \in \text{cone}(H)$ arbitrary. Let $B := \{a_1, \ldots, a_s\} \subseteq H$ be the vectors

---

* It is easy to see that the c-maximum solutions are exactly the lexicographically maximal ones, if we order the variables in the decreasing order of c. In Chandrasekaran and Tamir's terminology our proof means that strong Hilbert bases are exactly those in which lexicographically maximal solutions *with respect to a specific order* are Hilbert bases.

which belong to a positive variable β_i in an optimal basic solution for the linear program $\sum_{h\in H} x(h)h = b, \quad x \geq 0, \quad \max \sum_{h\in H} x(h)c(h)$. (The maximum exists because of the pointedness of cone(H).)

B must be a Hilbert basis, because if it were not, then (3.1) holds, that is

$$b = \varepsilon \sum_{h\in H} \alpha(h)h + \sum_{i=1}^{s}(\beta_i - \varepsilon)a_i. \tag{3.3}$$

Because of uncrossability, (3.2) also holds. If ε is sufficiently small, (3.3) also expresses b as a nonnegative combination, and by (3.2), has bigger objective value than $\sum_{i=1}^{s} \beta_i c(a_i)$, a contradiction with the choice of B.

Q.E.D.

The above proof implies that cone(H), where H is an uncrossable Hilbert basis, has a "triangulation" with cones generated by independent Hilbert bases $B \subseteq H$. A *triangulation* of the cone C is a set of cones with linearly independent extreme rays whose union is C, and the intersection of any two of which is a smaller dimensional cone.

4. Combinatorial examples

We have now finished the study of Hilbert bases in general. In this section we sketch some applications, in the following section we make some algorithmic and other remarks. In these two sections the reader will have to be satisfied with a short summary, and sketched proofs because of the lack of place (and time). I hope to write more details in a forthcoming paper.

Hilbert basis constitute an equivalent algebraic language to study TDI systems. A system of inequalities is called *totally dual integral* (or TDI), if any inequality which is their consequence and has an integer coefficient vector, arises as their non-negative integer combination. The first, basic results about TDI systems were proved by Giles and Pulleyblank(1979), Edmonds and Giles (1984), and Schrijver (1981). The translation between TDI systems and Hilbert bases is provided by the following observation of Giles and Pulleyblank (1979) see also Schrijver (1986):

Theorem 4.1 a. *The system of inequalities $Ax \leq b$ (A and b are integral) is TDI if and only if for each face F of the polyhedron $\{x : Ax \leq b\}$, the rows of A which are active in F form a Hilbert basis.*
b. *The rows of the integer matrix A form a Hilbert basis if and only if $Ax \leq 0$ is TDI.*

A row of a is called *active* in a face (or for an objective function), if it is satisfied with equality by every vector in the face (which is the set of optimal solutions for the given objective function).

Many combinatorial applications of Hilbert bases arise from well-known TDI systems of Combinatorial Optimization, (and as we shall see below, many others arise in a different way). It is an immediate conequence of Theorem 2.1 (through Theorem 4.1) that *for n-dimensional TDI systems there always exists an integer optimal dual solution with at most* $2n-2$ *nonzero variables.* We list now some Hilbert bases, where we know something better than the bound $2n-2$.

I. Totally unimodular and uncrossable systems

For the Hilbert bases corresponding to many TDI systems all the conjectures we presented hold trivially. Two big classes of them whill let us avoid listing them one by one: integer linear programs with totally unimodular constraint matrices; problems where for example uncrossing procedures lead to a "triangular basic" dual solution. An example to the latter: matroid polyhedra.

II. Perfect Graphs and Matchings

The following Lemma extracts the essence of Fulkerson's proof (1972) of the Plu-perfect Graph Theorem:

Lemma *Let* $P := \{x : Ax \leq b\}$ *be full dimensional (for simplicity), and let* $ax \leq \beta$ *be an inequality in the system* $Ax \leq b$. *Suppose furthermore, that* $ax \geq \beta - 1$ *is valid for every* $x \in P$. *If for an objective function* c $ax \leq \beta$ *is active, then there exists a dual solution to the linear program* $Ax \leq b$, $\max cx$ *where the dual variable corresponding to the row* a *is at least* 1. .

Theorem 4.2 *The "active rows" (for an arbitrary objective function) of clique polyhedra of perfect graphs and matching polyhedra form a strong Hilbert basis.*

Proof. (Sketch) All the inequalities describing the clique polyhedron of perfect graphs are of the form $ax \leq 1$, and $ax \geq 0$ is valid for them. The Lemma states that such inequalities are "strong" elements of the cone of active rows: it follows that the active rows form a strong Hilbert basis.

For matching polyhedra, inequalities of the type $x_i \geq 0$ and $x(\delta(v)) \leq 1$ satisfy the conditions of the Lemma. So whenever such an inequality is active, it is a strong element. If there are no such inequalities, then it can be shown that one of the "odd set inequalities" is strong. Q.E.D.

For the case of perfect graphs, the above proof is just an equivalent way of presenting Cook, Fonlupt and Schrijver's (1986) proof of Conjecture A.

Corollary *Conjectures A, B, C, D hold for the Hilbert bases in the above theorem*

III. MATROID BASES

It follows from the matroid partition theorem that the (incidence vectors of the) bases of an arbitrary matroid form a Hilbert basis (whose lattice is not $\mathbb{Z}^n$). A. Frank (1984) has proved that *bases of uniform matroids* (the set of all k-tuples of an arbitrary set) form a Hilbert basis for which Conjecture A holds. É Tardos (1984) gave a simple proof that reduced the problem in one step to a smaller dimensional one, actually with a natural choice of a "strong element". This leads to the following:

Theorem 4.3 *The set of all k-tuples of an arbitrary set forms a strong Hilbert basis, where every element is strong (in every face containing it).*

Corollary *Conjectures A, B, C, D hold for the set of all k-tuples of an arbitrary set.*

This is the only result I know about the conjectures on matroid bases.

IV. ARBORESCENCES

It follows from Edmonds' (1967) rooted arborescence theorem that arborescences (as edge-sets) constitute a Hilbert basis.

From Pevsner's (198?) algorithmic considerations one can extract the following theorem:

Theorem 4.4 : *Let G be a directed graph with a root $r \in V(G)$, and w be a vector in the cone C generated by the arborescences of G rooted in r.*
a. *If $w - a \notin C$ for some arborescence a, then there exists an arborescence b such that $w - b \in C$ and the dimension of the minimal face containing $w - b$ is less than that of w.*
b. *There exist two arborescences a and b rooted in r and a non-negative integer α such that $w' := w - (\alpha a + b) \in C$, and the dimension of the minimal face containing w' is less than that of w.*

It is easy to prove Theorem 4.4a using Lovász's proof of Edmonds' theorem, and b is an easy corollary of a.

Theorem 4.4b implies a better bound than the bounds known in general: using two vectors, the dimension can be decreased by one; if Conjecture A is proved for graphs where the linear rank of the arborescences is c, we get that for every w there exists a solution with at most $2(r-c)+c = 2r-c$ positive coefficients, where r is the rank of the arborescences; we get immediately the bound $2r-3$ (because of Theorem 2.2), that can probably be considerably improved with an analysis of small graphs. We would find it more interesting however, and we hope too, that the strong property exhibited

in Theorem 4.4 brings us nearer to the conjectures in this case.

The Conjectures for the "polar" problem (minimum weight rooted arborescence maximum cut packing) are trivial, belong to I.

V. ODD CUTS

The following example was suggested by Les Trotter (1987): *For every objective function, the active rows of T-join polyhedra form a Hilbert basis, if the non-negativity constraints are written in the form $2x_i \geq 0$.* This follows from Seymour's integer minimax theorem (1981a) about minimum T-joins and maximum T-cut packings in graphs with "bipartite" weightings. The lattice generated by the odd cuts and the doubles of the unit vectors is clearly *the set of bipartite weightings,* and not the set of all integer vectors: this case does not fit into the usual definition of TDI-ness, which supposes that the considered lattice is the set of all integers.

Some special cases such as planar graphs are strong Hilbert bases, and thus the conjectures hold for them.

VI. MULTICOMMODITY FLOWS

It is a basic question about graphs whether *the existence of a fractional multicommodity flow implies the existence of an integer one* for arbitrary demands and capacities, see Seymour (1981b), Karzanov (1987), Sebő (1990). Such graphs are called *routing.*

Given a graph G let the *multicommodity cone* of G be the cone generated by the vectors $v_{C,f}$ and $2e_i$, where $C \subseteq E(G)$ is a cycle, $f \in C$, and $v_{C,f}(e)$ is equal to -1 if $e = f$, to 1 if $e \in C \backslash \{f\}$, and 0 otherwise; the e_i-s are the unit vectors on the edges. The extreme rays defined here will be denoted by MH(G). The above mentioned question, and the papers refered to investigate the problem of characterizing when MH(G) or some subcone of it is a Hilbert basis.

It is easy to see that a graph is *routing* if and only if MH(G) is a Hilbert basis. Routing graphs include Seymour's (1981b) class of "cycling" graphs. More generally, all this can be defined in terms of binary matroids. The characterization of routing matroids is open.

We do not require that the generated lattice is the set of all integer vectors: if we require that, it follows from Seymour (1981b) that MH(G) is a Hilbert basis if and only if G is series-parallel. MH(G) generates the lattice of Eulerian weightings. The MH(G) of certain graphs (or matroids), among them that of planar graphs, turns out to be a strong Hilbert basis. Consequently Conjectures A, B, C, D hold for these cases.

The list of the examples seems to be unbounded, actually every TDI system gives an example. On the other hand the list of the solved cases of the conjectures is for the moment poor.

5. Other remarks

I. Testing membership

We first prove that *it is coNP-complete to decide whether a given vector is in the Hilbert basis of a cone given by defining inequalities*, a fact proved first by É. Tardos (1987) with a reduction to the maximum stable set problem of a graph.

Theorem 5.1 *Given two vectors $a, h \in \mathbb{Z}^n$, it is coNP-complete to decide whether h is in the Hilbert basis of the cone $\{x : ax = 0, x \geq 0\}$*

Proof. Let $a_1, \ldots, a_{n-2}$ be an instance of PARTITION (see Garey-Johnson (1979), and $a_{n-1} := a_n := -1/2(a_1 + \ldots + a_{n-2})$, $a := (a_1, \ldots, a_n)$; let h be the n-dimensional all 1 vector. Clearly, $C := \{x : ax = 0, x \geq 0\}$ is a pointed cone, $h \in C$, and h is in the Hilbert basis of C if and only if the answer to the given instance of PARTITION is no.

Q.E.D.

II. Testing TDI-ness, and integral dual solutions

Next we show that *Conjecture B contains Cook, Lovász and Schrijver's result (1984) on testing TDI-ness in fix dimension.* We do not have to use Lenstra's integer programming algorithm in the test. The key-observation is the following:

Theorem 5.2 *Suppose Conjecture B is true, and $H \subseteq \mathbb{Z}^n$ is full dimensional (and generates a pointed cone). Let $H_1, \ldots, H_p \subseteq H$ be the list of those n-element independent subsets of H whose determinant is 1. Then H is a Hilbert basis if and only if* $\text{cone}(H) = \bigcup_{i=1}^{p} \text{cone}(H_i)$.

Proof. The only if part is just Conjecture B. The if part is trivial, since by the constraint $w \in \text{cone}(H)$ is in some of the $\text{cone}(H_i)$-s, and those are Hilbert bases.

Q.E.D.

Let H be like in the theorem, and let n be fixed. (It is easy to see from Theorem 4.1 that in order to check TDI-ness it is enough to test whether a given set of vectors is a Hilbert basis; the assumptions of Theorem 5.2 on H do not restrict the generality, see Schrijver (1986)). Since n is fixed, the facets of $C_0 := \text{cone}(H)$ can be determined in polynomial time. Let now $H_1, \ldots, H_p \subseteq H$ be the list of those n-element independent subsets of H whose determinant is 1. Since n is fixed, this list, and the facets of $C_i := \text{cone}(H_i)$ $(i = 1, \ldots, p)$ can be determined in polynomial time. Let $b_1, \ldots, b_q \in \text{cone}(H)$ be the list of all vectors that generate a one-dimensional cone (half line) which is the intersection of n linearly independent facets from among the facets of $C_0, \ldots, C_p$. Since n is fixed, q is still a polynomial of $|H|$, and can be computed in polynomial time.

Now clearly, $\text{cone}(H) \neq \bigcup_{i=1}^{p} \text{cone}(H_i)$ *if and only if there exists a linearly independent subset* $\{c_1, \ldots, c_n\} \subseteq \{b_1, \ldots, b_q\}$ *such that no inner (non-facet) point of* $\text{cone}(c_1, \ldots, c_n)$ *lies in* $\bigcup_{i=1}^{p} \text{cone}(H_i)$. Thus we "only" have to test for all $\{c_1, \ldots, c_n\} \subseteq \{b_1, \ldots, b_q\}$ whether $c_1 + \ldots + c_n$ (this is an inner point) is contained in at least one of the cones C_i $(i = 1, \ldots, p)$. If yes, then we can conclude $\text{cone}(H) = \bigcup_{i=1}^{p} \text{cone}(H_i)$, and H is a Hilbert basis by Theorem 5.3. If $c_1 + \ldots + c_n$ is contained in none of the C_i $(i = 1, \ldots, p)$ for some $\{c_1, \ldots, c_n\} \subseteq \{b_1, \ldots, b_q\}$, then clearly $\text{cone}(H) \neq \bigcup_{i=1}^{p} \text{cone}(H_i)$, and according to Theorem 5.2 H is not a Hilbert basis. Q.E.D.

For TDI systems, we are also interested in finding an integer dual solution, and if the conjectures hold, a dual solution whose positive variables belong to linearly independent constraints forming a Hilbert basis. If our TDI system is given explicitly by a system of linear inequalities and Conjecture D holds, even the latter task can be solved easily with the help of Theorem 4.1.

For linear programming problems given by a separation oracle (for the definition see Grötschel Lovász, Schrijver (1987)) we must suppose that the minimal system describing the corresponding polyhedron is TDI, and that for a cone given by a separation oracle, and whose extreme rays constitute a Hilbert basis, $c(h)$ in Conjecture D can be determined in polynomial time. (For most of the TDI systems for which the Conjectures are known, these conditions are satisfied.) With the proof technique used by Grötschel, Lovász and Schrijver (1987) (6.5.14), our problem is reduced to writing a vector w of a cone given by a separation oracle as a non-negative c-maximum linear combination of the extreme rays. (By the proof of Theorem 3.2 every independent subset of a c-optimal solution is a Hilbert basis.)

Compared to (6.5.14) here we have the additional task of optimizing the function c. However, given a separation oracle for the cone $\{yA : y \geq 0\}$, a separation oracle for the polyhedron $\{yA = w : y \geq 0\}$ can be determined in polynomial time, whence the function c can be optimized in polynomial time. (We used the equivalence of optimization and separation several times, see Grötschel, Lovász and Schrijver' book.)

III. Structure

We prove here some weakenings and of Conjecture B, and state a reformulation.

Note first that there exists at least one basis with the property claimed in Conjecture B (Gerards and Sebő (1987)), implying "local strong unimodularity" for totally dual integral systems, that is the existence of active rows whose determinant is 1.

Next we note that *every cone has a finite Hilbert generating system for which Conjectures A, B, C are true*, see Remark 1 after Theorem 2.1, but the number of redundant elements in this construction might be huge. Of course, if we start with a covering of $\text{cone}(H)$ by the cones of Conjecture B, then there there is no redundant

element. It would be interesting to prove the existence of a Hilbert generating system for which the conjectures hold and does not have many redundant elements.

par(H) is a Hilbert generating system which has many redundant elements, but the conjectures are still interesting and open for it. Liu and Trotter (1990) have put the interesting question of giving a proof at least if "1" is replaced by some bigger, but not very big number in the definition of "par", and they have some results in this direction.

Finally, we sketch our results on Hilbert kernels: they are based on the recognition that for any property of Hilbert bases, only the linear dependencies between Hilbert-basis elements play a role. A linear subspace $S \subseteq \mathbf{R}^k$ is called *Hilbert kernel*, if there exists an $n \times k$ matrix H whose columns constitute a Hilbert basis (of a pointed cone), and $S := \{x : Hx = 0\}$. A great advantage of Hilbert kernels is that they neglect non-essential properties of Hilbert bases, for example their definition does not depend on whether we suppose $\mathrm{lat}(H) = \mathbb{Z}^n$ or not. We present the following characterization of Hilbert kernels without proof, noting that the proof is not difficult.

Theorem 5.3 *Let $S \subseteq \mathbb{Z}^n$ be a linear subspace. The following statements are equivalent:*

(i) S is a Hilbert kernel

(ii) For every $x \in S$ there exists $y \in S$ so that $y \equiv x$ and $y < 1$

(iii) For every $x \in S$ there exists $y \in S$, y integer and $y \leq \lceil x \rceil$

(iv) $H(S) \cup \{e_1, \ldots, e_k\}$ is a Hilbert basis, where $H(S)$ is a Hilbert basis of S, and e_i $(i = 1, \ldots, k)$ is the vector whose i-th coordinate is 1, and the others are 0.

Let us reformulate Conjecture B in terms of Hilbert kernels (Conjectures A, C, D can be reformulated similarly). The proof of the equivalence is easy.

CONJECTURE E *If S is a Hilbert-kernel, then for every $x \in \mathbb{R}^n, x \geq 0$ there exists $y \in S$, such that $z \in S$, z_i integer for values i with $x_i = y_i$, implies $z \in \mathbb{Z}^n$.*

The coefficients of the linear equations satisfied by an n-dimensional Hilbert basis of cardinality k form a $k-n$ dimensional subspace of $\mathbb{R}^k$. It is convenient to represent Hilbert-kernels with a basis of $S \cap \mathbb{Z}^k$, $k-n$ linearly independent k-dimensional vectors. The case $k = n+1$ corresponds to 1-dimensional Hilbert-kernels. From Theorem 5.3 it is easy to see that 1-dimensional Hilbert kernels are exactly the lines generated by vectors all positive components of which are 1. Conjecture E is an immidiate consequence, providing a new proof of Theorem 2.3.

At this point, the relation of the different conjectures may appear chaotic. Let us finally summarize the easy implications by the following scheme:

CONJECTURE D $\Longrightarrow$ CONJECTURE B $\Longleftrightarrow$ CONJECTURE C $\Longleftrightarrow$ CONJECTURE E $\Longrightarrow$ CONJECTURE A.

We do not know anything about the other implications.

Acknowledgment: I owe a lot to Bill Cook, Bert Gerards and Éva Tardos: most of the above results were first written down in letters to them, and would not have been proved without their permanent interest and encouragement. Many thanks are due to Brahim Chaourar and Kazuo Murota for various suggestions.

References:

Chandrasekaran, Tamir (1984), On the integrality of an extreme solution to pluperfect graph and balanced systems, Operations Research Letters, **3**,4, 215–218 137–145

W. Cook, J. Fonlupt, A. Schrijver (1986), An integer analogue of Carathéodory's theorem, Journal of Combinatorial theory (B) **40** (1986) 63–70

W. Cook, L. Lovász, A. Schrijver (1984) A polynomial-time test for total dual integrality in fixed dimension, Mathematical Programming Study,**22** (1984), 64–69.

W. Cunningham (1987) Testing membership in matroid polyhedra, J. Combinatorial Theory B 36, 161-188

J. Edmonds (1967) Edge-disjoint branchings, in: Combinatorial algorithms, Academic Press, New York

J. Edmonds (1970) Submodular functions, matroids and certain polyhedra, in: Combinatorial Structures and their Applications, Gordon and Breach, New York, pp 69–87

J. Edmonds R. Giles (1984) Total dual integrality of linear inequality systems, in: Progress in Combinatorial Optimization, Jubilee Conference, W.R. Pulleyblank ed., Academic Press, Toronto

A. Frank (1984) private communication

A. Frank (1987) Graph connectivity and network flows, to appear in the Handbook of Combinatorics, R. Graham, M. Grötschel, L. Lovász eds.

D.R. Fulkerson, Antiblocking Polyhedra, J. of Comb. Theory,**12** (B), pp 50–71

M.R. Garey, and D.S. Johnson (1979), Computers and Intractability: a Guide to the Theory of NP-completeness, Freeman, San Francisco

A. Gerards, A. Sebő (1987), Total dual integrality implies local strong unimodularity, Mathematical Programming, **38**, 69–73

Giles, Pulleyblank (1979), Total dual integrality and integer polyhedra, Linear algebra and its applications, 25, 191–196

Grötschel, Lovász, Schrijver (1987) The ellipsoid method and combinatorial optimization, Springer, Heidelberg

A.V. Karzanov (1987)Half integral five-terminus flows, Discrete Applied Mathematics, 18, 263–278

J. Liu and L. Trotter (1990) private communication

L. Lovász (1976) On two minimax theorems in graph, J. Comb Theory, B, **2**, 96–103

L. Lovász (1987) Matching Theory, Lecture at the seventh Hungarian Colloquium on Combinatorics, Finite and Infinite sets, Eger, July 5-10

A. Hoffmann and R. Oppenheim (1978), Local unimodularity in the matching polytope, Annals of Discrete Mathematics, 2, 201–209

P. A. Pevsner (198?) Effektivnyi algoritm upakovki vetvlenii vo vzveshennom grafe (in Russian)

A. Schrijver (1981), On total dual integrality, Linear algebra and its applications, **38**, 27–32

A. Schrijver (1986), Theory of linear an integer programming, Wiley, Chichester

A. Sebő (1990) The cographic multiflow problem: an epilogue, to appear in DIMACS (W. Cook and P. Seymour eds.)

P.D. Seymour (1981a), On odd cuts and plane multicommodity flows, Proc. London Math. Soc. **(3) 42**, 1981, 178–192

P.D. Seymour (1981b), Matroids and multicommodity flows, European Journal of Combinatorics **(2)** , 257–290

É. Tardos (1984), (1987) private communication

L. Trotter (1987) private communication

Stability Critical Graphs and Even Subdivisions of K_4

E. C. Sewell and L. E. Trotter, Jr.*
School of ORIE
Cornell University
Ithaca, New York 14853

Abstract

A graph is stability critical (α-critical) if the removal of any edge increases the stability number. We give an affirmative answer to a question raised by Chvátal, namely, that every connected, critical graph that is neither K_2 nor an odd cycle contains an even subdivision of K_4.

All graphs in this paper are assumed to be finite, simple and undirected. For graph $G = (V, E)$, we also denote $V(G) = V$ and $E(G) = E$. A set of mutually nonadjacent nodes in a graph G is called a *stable (independent) set.* A *maximum stable set (MSS)* is a stable set of maximum cardinality. The *stability number* of G, denoted by $\alpha(G)$, is the cardinality of a maximum stable set in G. A stable set S *saturates* G if $|S| = \alpha(G)$. The degree of a node v in G is denoted by $d(G, v)$ (whenever G is clear from the context, it will be suppressed from the notation). A node with degree equal to zero is said to be *isolated.* K_n is the complete graph on n nodes. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$, denoted $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. If $W \subseteq V$, then $G[W]$ denotes the subgraph *induced* by W; i.e., $G[W]$ has node set W and two nodes are adjacent in $G[W]$ if and only if they are adjacent in G. If $v \in V$, then $G - v$ will also be used to denote $G[V \setminus \{v\}]$. If $(u, v) \in E$, then $G - (u, v)$ denotes the subgraph $(V, E \setminus \{(u, v)\})$.

An edge of G is said to be *critical* if its deletion increases the stability number. G is *α-critical* if every edge of G is critical. Throughout this paper, *critical* will always mean α-critical. If (v, w) is a critical edge of G, then there is a MSS in G that contains v and there is a MSS that contains w. This follows by considering a MSS S in $G - (v, w)$. Since (v, w) is critical, S must have cardinality $\alpha(G) + 1$, which then implies that $v, w \in S$. Thus, $S \setminus \{v\}$ and $S \setminus \{w\}$ are MSS's in G that contain w and v, respectively. If G is a critical graph and $(v, w) \in E$ with $d(v) + d(w) > 2$, then there is a MSS in G that contains neither v nor w. To see this, assume, without loss of generality, that $(u, v) \in E(G)$ with $u \neq w$ and let S be a MSS in $G - (u, v)$. Then $S \setminus \{v\}$ is the desired MSS in G. Finally, if G is critical and $v \in V$ with $d(v) = 1$, then v and its neighbor form a component of G, since every MSS in G contains either v or its neighbor.

The number $|V| - 2\alpha(G)$, denoted by $\delta(G)$, plays an important role in the study of critical graphs, as demonstrated by the following theorem given by Hajnal in 1965 [3].

Theorem 1 (Hajnal) *If G is a critical graph with no isolated nodes, then $d(v) \leq \delta(G) + 1$ $\forall v \in V$.*

This theorem is useful in characterizing critical graphs with small values of $\delta(\cdot)$. Let Γ^δ be the set of all critical graphs with $\delta(G) = \delta$ and let Γ_c^δ be the set of all connected graphs in Γ^δ. If $G \in \Gamma_c^0$, then every node of G has degree at most one, which implies that G is K_2. If $G \in \Gamma_c^1$, then every node of G has degree at most two. Since G is connected, G must be either a simple path or a cycle. But simple paths and even cycles are not critical, so G must be an odd cycle. A *subdivision* of a graph is obtained by replacing its edges by simple paths, i.e., by inserting new nodes of degree two into the edges. An *even subdivision* results when the number of new nodes inserted into each edge is even. Hence, Γ_c^1 consists of even subdivisions of K_3. The situation for Γ_c^2 is more complex, but Andrásfai [1] established the following theorem in 1967.

Theorem 2 (Andrásfai) *If $G \in \Gamma_c^2$, then G is an even subdivision of K_4.*

*Research partially supported by NSF Grants ECS8504077 and DDM-8814644

In 1978 Lovász [5] established that for each fixed value of δ there is a finite set of graphs (a finite "basis") such that every graph in Γ_c^δ is an even subdivision of one of these basis graphs. The preceding discussion together with Theorem 2 imply that K_2 is the basis for Γ_c^0(in fact, K_2 is the only graph in Γ_c^0), K_3 is the basis for Γ_c^1 and K_4 is the basis for Γ_c^2. Furthermore, in [6] it is shown that there is a finite basis for Γ_c^δ using the more general operation defined in the following theorem. (The basis for Γ_c^3 is given explicitly in [6].)

Theorem 3 (Lovász and Plummer) *Let G be a critical graph and x a node of degree two in G. Let y and z be the neighbors of x. If y and z are adjacent, then $\{x, y, z\}$ forms a component of G. If y and z are not adjacent, then no node different from x is adjacent to both of them and furthermore, if the edges (x, y) and (x, z) are contracted, the resulting graph G' is critical with $\alpha(G') = \alpha(G) - 1$ and $\delta(G') = \delta(G)$. Conversely, suppose G' is a critical graph and w is any node of G'. Split w into two nodes y and z, each of degree at least one, create a new node x and connect it to both y and z. Then the resulting graph G is critical with $\delta(G) = \delta(G')$.*

A subgraph H of G is said to be a *δ-subgraph* of G if H is critical, $V(H) = V(G)$, $\alpha(H) = \alpha(G)$ (hence $\delta(H) = \delta(G)$) and H does not contain any isolated nodes. In 1975 Surányi [7] proved the following two results concerning δ-subgraphs.

Lemma 4 (Surányi) *Let G be a critical graph and $(v, w) \in E(G)$. If H is a δ-subgraph of $G - v$, then $d(H, w) = d(G, w) - 1$.*

Theorem 5 (Surányi) *If G is a critical graph without isolated nodes and $v \in V$ with $d(v) > 1$, then there exists a δ-subgraph of $G - v$.*

Harary and Plummer [4] showed that every critical graph with $\delta(\cdot) \geq 1$ contains an odd cycle, i.e., an even subdivision of the basis graph for Γ_c^1. In 1975 Chvátal [2] proved that every connected, critical graph with $\delta(\cdot) \geq 2$ contains a subdivision of K_4 and he posed the question of whether every connected, critical graph with $\delta(\cdot) \geq 2$ must contain an even subdivision of K_4, i.e., an even subdivision of the basis graph for Γ_c^2. The following theorem establishes an affirmative answer to this question.

Theorem 6 *If $G = (V, E)$ is a connected, critical graph with $\delta(G) \geq 2$, then G contains an even subdivision of K_4.*

Proof. We first note that we may assume with no loss of generality that $d(G, v) \geq 3$, $\forall v \in V$. To see this, note that G is connected, so G contains no isolated nodes. Furthermore, if G had a node of degree one, then G could only consist of a single edge, contradicting $\delta(G) \geq 2$. Finally, if $d(G, x) = 2$, suppose $(y, x), (z, x) \in E$ with $y \neq z$. Then Theorem 3 implies $(y, z) \notin E$. Thus, again by Theorem 3, we can remove x and identify y and z to obtain a connected, critical graph G' with $\delta(G') \geq 2$. It is not difficult to see that G contains an even subdivision of K_4 provided G' does, so we replace G by G'. Repeated application of this argument allows us to assume $d(G, v) \geq 3$, $\forall v \in V$.

We may further assume that no proper subgraph of G satisfies the assumptions of the theorem, i.e., that G is *minimal* with respect to the stipulations connected, critical and $\delta(G) \geq 2$ (else we could replace G by such a subgraph and proceed with the proof). It follows that we need only consider cubic graphs, for if $d(G, w) > 3$ with $(v, w) \in E$, then let H be a δ-subgraph of $G - v$ (see Theorem 5). By Lemma 4, $d(H, w) \geq 3$. If H' is the component of H containing w, then H' is connected and critical with $\delta(H') \geq d(H', w) - 1 \geq 2$ (see Theorem 1), contradicting the minimality of G.

The proof thus reduces to showing that any connected, critical, cubic, minimal graph $G = (V, E)$ contains an even subdivision of K_4. We denote $n = |V|$, $\alpha = \alpha(G)$ and $\delta = \delta(G)$.

Claim 1 *Suppose $v \in V$ and H_v is a δ-subgraph of $G - v$. Let $E_v = E \setminus E(H_v)$. Then:*
(i) $\alpha(H_v) = \alpha(G)$ and $\delta(H_v) = \delta(G) - 1$.
(ii) H_v consists of isolated edges and $\delta - 1$ odd cycles.
(iii) If $(v, w) \in E$, then w is contained in an odd cycle in H_v.
(iv) $\alpha(G - v - e) = \alpha$ $\forall e \in E_v$.

(v) If $(v,w) \in E$, *then* $E_v \cap E_w = \{(v,w)\}$.
(vi) $\forall w \in V$, *some edge incident to* w *is in* E_v.
(vii) Let $C_1, \ldots, C_{\delta-1}$ *be the odd cycles in* H_v *and* $e_1, \ldots, e_s$ *be the isolated edges. If* I *is a stable set in* G *with* $|I| = \alpha$ *and* $v \notin I$, *then* I *saturates* $C_1, \ldots, C_{\delta-1}$ *and* $e_1, \ldots, e_s$.

Proof.
(i) Since G is critical, there exists a MSS which does not include v. Therefore, $\alpha(H_v) = \alpha(G-v) = \alpha$ and $\delta(H_v) = (n-1) - 2\alpha(H_v) = n - 1 - 2\alpha = \delta - 1$.
(ii) H_v is a δ-subgraph, so there are no isolated nodes. Each component of H_v must be connected and critical; moreover, minimality of G forces $\delta(\cdot) \leq 1$ for each component of H_v. Therefore, each component is an isolated edge or an odd cycle (see the discussion preceding Theorem 2). Furthermore, $\delta(H_v)$ equals the sum of $\delta(\cdot)$ for each component, so H_v contains $\delta - 1$ odd cycles.
(iii) This follows from Lemma 4 and (ii).
(iv) Suppose $e \in E_v$. Then $\alpha(G-v-e) \geq \alpha(G-v) = \alpha$ and $\alpha(G-v-e) \leq \alpha(H_v) = \alpha$.
(v) Suppose $(v,w) \in E$ and $e \in E_v \cap E_w$. It is easy to see that $(v,w) \in E_v \cap E_w$, so suppose that $e \neq (v,w)$. Let I be a stable set in $G-e$ with $|I| = \alpha + 1$. Clearly, I cannot contain both v and w, so without loss of generality assume $v \notin I$. Then I is a stable set in $G-v-e$ with $|I| = \alpha + 1$, and thus I is a stable set in H_v, which is a contradiction to (i).
(vi) Let $w \in V$. If no edge incident to w is in E_v, then w has degree three in H_v, contradicting (ii).
(vii) Suppose I is a stable set in G with $|I| = \alpha$ and $v \notin I$. Then I is a stable set in H_v with $|I| = \alpha(H_v)$. Therefore, I induces a MSS in each component of H_v; i.e., I saturates $C_1, \ldots, C_{\delta-1}$ and $e_1, \ldots, e_s$. □

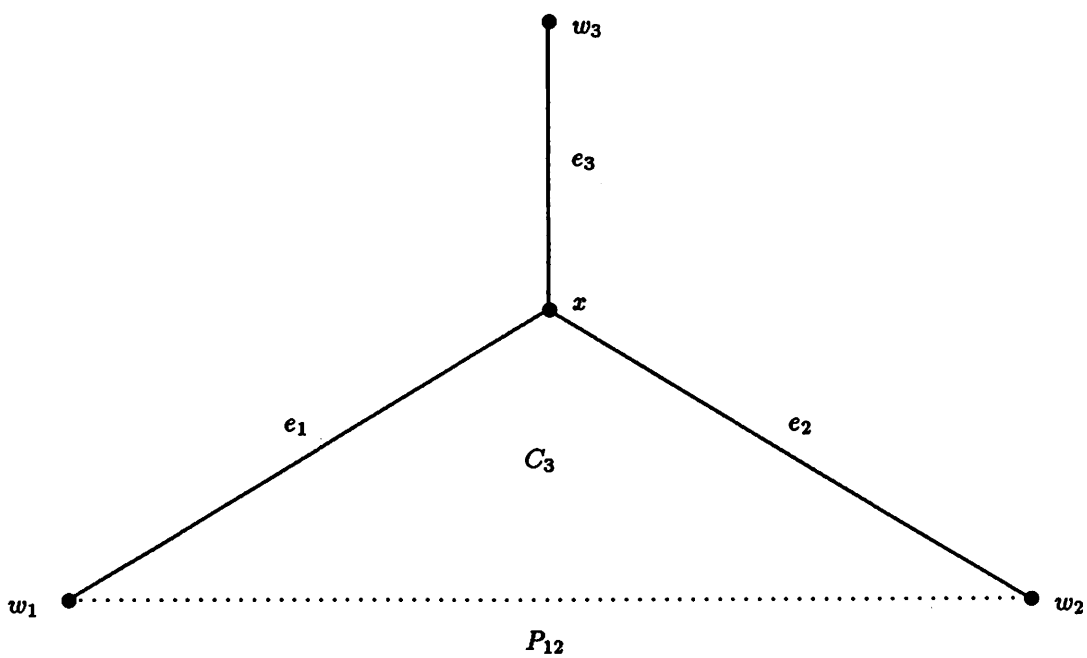


Figure 1: The structure of C_3.

Now let $x \in V$ and let $e_1 = (x, w_1)$, $e_2 = (x, w_2)$, $e_3 = (x, w_3)$ be the three edges incident to x. Let H_x, H_1, H_2 and H_3 be δ-subgraphs of $G-x$, $G-w_1$, $G-w_2$ and $G-w_3$, respectively. Let C_i be the odd cycle in H_i that contains x, for $i = 1,2,3$. Let P_{ij} be the path from w_i to w_j on $C_k \setminus \{x\}$, where $1 \leq i,j,k \leq 3$ are distinct indices. Obviously, P_{ij} has odd edge-length. Note that P_{ij} and P_{ji} have the same underlying undirected path. Now we want to show that $C_1 \cup C_2 \cup C_3$ contains an even subdivision of K_4. The main difficulty is that the P_{ij}'s may intersect. The remainder of the proof concentrates on finding nonintersecting subpaths of the P_{ij}'s that are of the correct parity to form an even subdivision of

K_4. See Figure 1.

In our subsequent development we let I_i be a stable set in $G - e_i$ with $|I_i| = \alpha + 1$, for $i = 1, 2, 3$.

Claim 2 *The nodes of P_{ij} alternate between I_i and I_j. (See Figure 2.)*

Proof. It is easy to see that $|I_i| = \alpha + 1$, $w_k \notin I_i$ and $\alpha(H_k) = \alpha$ imply that $|I_i \cap V(C_k)| = (|V(C_k)| + 1)/2$. This implies the nodes on P_{ij} which are of even distance from w_i are in I_i. Similar reasoning holds for w_j and I_j. □

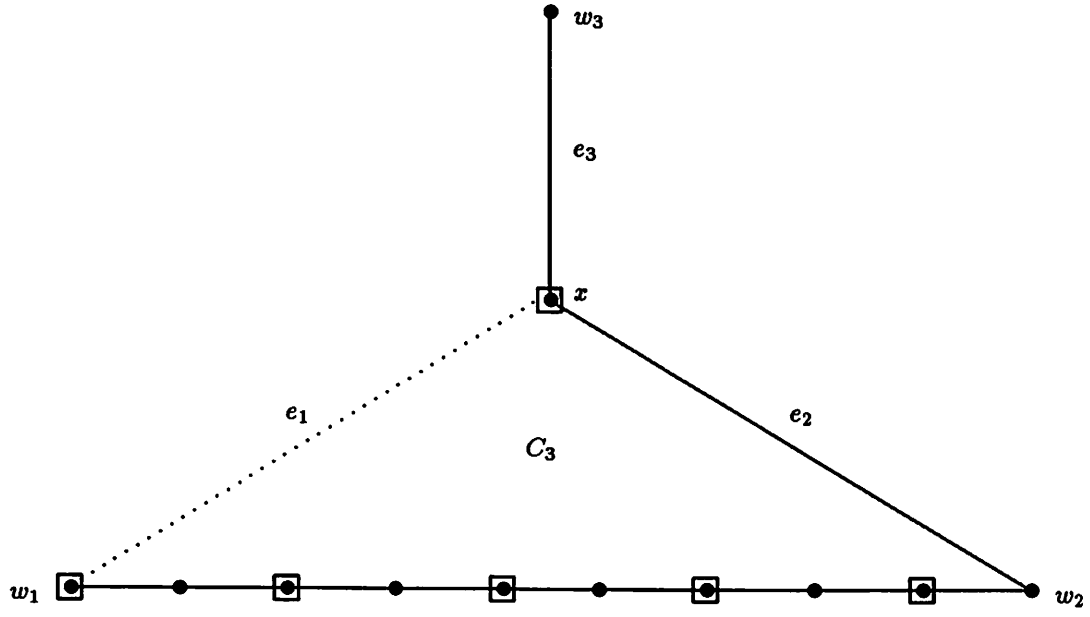


Figure 2: Distribution of I_1 on C_3.

Claim 3 *P_{ij} contains exactly one edge, call it e_{ij}, such that neither of its endnodes is in I_k. (See Figure 3.)*

Proof. The stable set $I_k \setminus \{w_k\}$ saturates C_k, so C_k contains exactly one edge such that both of its endnodes are not in I_k. This edge must lie on P_{ij} because $x \in I_k \setminus \{w_k\}$. □

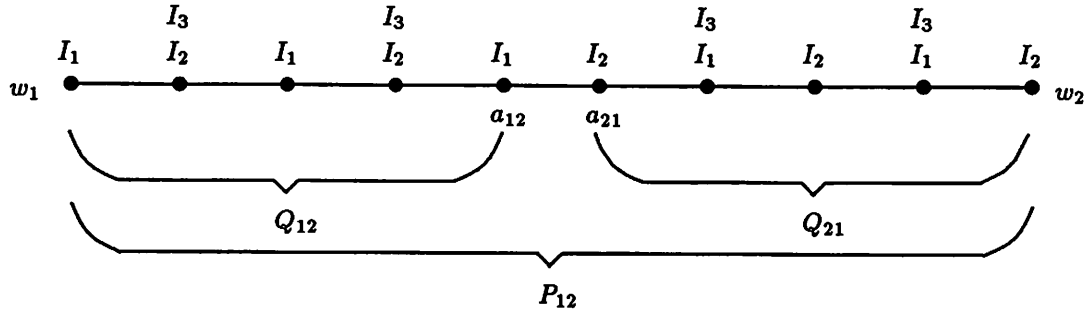


Figure 3: The alternation of the I_i's on Q_{12} and Q_{21}.

Let a_{ij} be the first node of e_{ij} encountered when traversing P_{ij} from w_i to w_j and let Q_{ij} be the subpath of P_{ij} from w_i to a_{ij}. Note that Q_{ij} and Q_{ji} are node-disjoint, whereas P_{ij} and P_{ji} specify the same underlying undirected path. Note also that Q_{ij} or Q_{ji} may consist of a single node.

Claim 4 *$I_j \cap Q_{ij} = I_k \cap Q_{ij}$ and $I_i \cap Q_{ji} = I_k \cap Q_{ji}$; i.e., the nodes of Q_{ij} alternate between I_i and $I_j \cap I_k$ with the first and last node of Q_{ij} in I_i (and the nodes of Q_{ji} alternate between I_j and $I_i \cap I_k$). (See Figure 3.)*

Proof. Let P be the path from a_{ij} to a_{ji} on $C_k \setminus \{e_{ij}\}$. Claim 3 implies that alternate nodes of P must be in I_k. Now, $x \in I_k$ implies $w_i \notin I_k$ and $w_j \notin I_k$. Furthermore, $a_{ij} \notin I_k$ and $a_{ji} \notin I_k$, by definition. Finally, by Claim 2, Q_{ij} alternates between I_i and I_j, and the result follows. □

We observe that the proof of Claim 4 also demonstrates that $a_{ij} \in I_i$, for any distinct index pair.

Claim 5 *The path P_{ik} contains no node of Q_{ji} (nor of Q_{jk}). (See Figure 4.)*

Proof. Since P_{ik} alternates between nodes of I_i and I_k, it contains no node of $I_i \cap I_k$ nor any node of $I_j \setminus (I_i \cup I_k)$. But Q_{ji} alternates between nodes in I_j and $I_i \cap I_k$. □

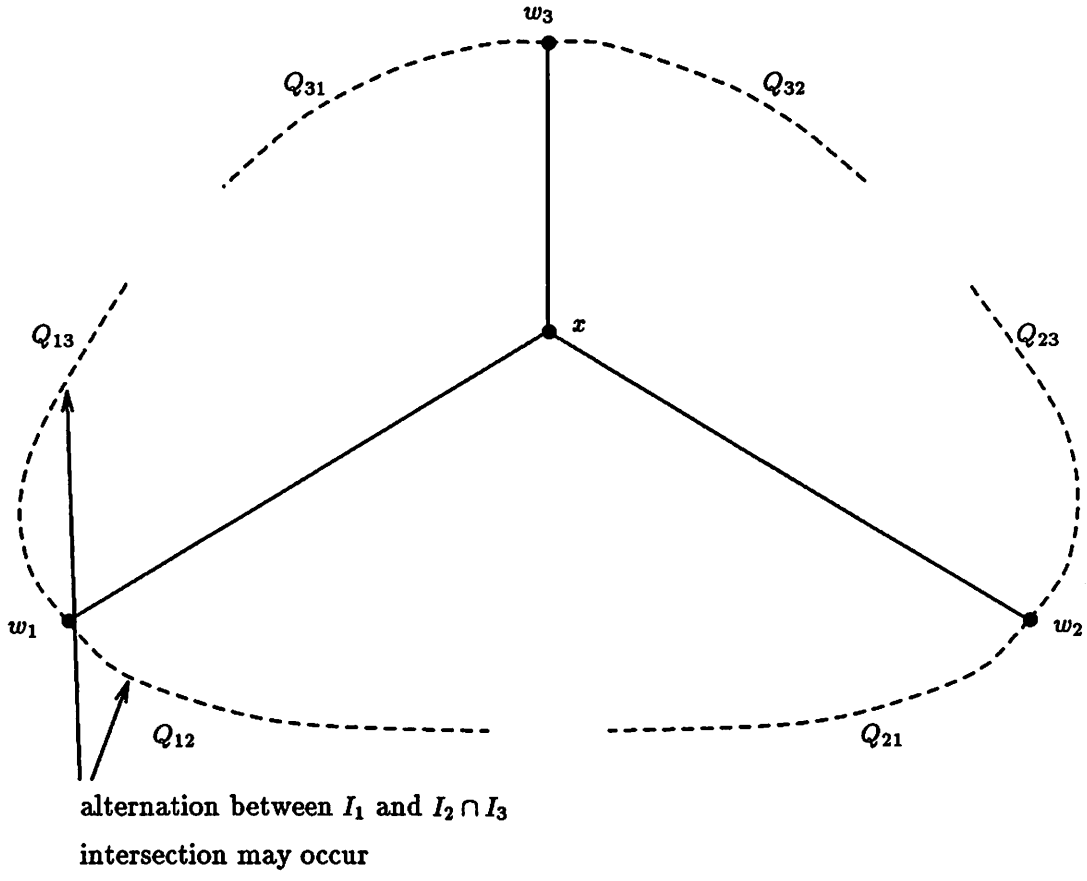


Figure 4: The structure of the Q_{ij}'s.

We now denote $E_x = E \setminus E(H_x)$ and $E_i = E \setminus E(H_i)$, for $i = 1, 2, 3$.

Claim 6 *Let the edges of P_{ij} be labeled $d_1, \ldots, d_r$. Then the edges with odd index are not in E_x and the edges with even index are in E_x; i.e., the edges of P_{ij} are alternately in and not in E_x, with the first and last edge not in E_x.*

Proof. Let $d_m = (u, v)$ be an edge of P_{ij} with m odd. (See Figure 5(a).) Suppose $d_m \in E_x$ and I is a stable set in $G - d_m$ with $|I| = \alpha + 1$. Then, by Claim 1(iv), $d_m \in E_x$ implies $x \in I$. Thus $w_k \notin I$, so Claim 1(vii) implies both $I \setminus \{u\}$ and $I \setminus \{v\}$ saturate C_k. A simple parity argument shows that this cannot happen, so $d_m \notin E_x$. Now suppose m is even and consider the three edges adjacent to v. (See Figure 5(b).) One of these edges is d_{m+1}, which is not in E_x by the above argument. Another one of the edges is in E_k and therefore not in E_x, by Claim 1(v). Hence, the remaining edge, d_m, must be in E_x by Claim 1(vi). □

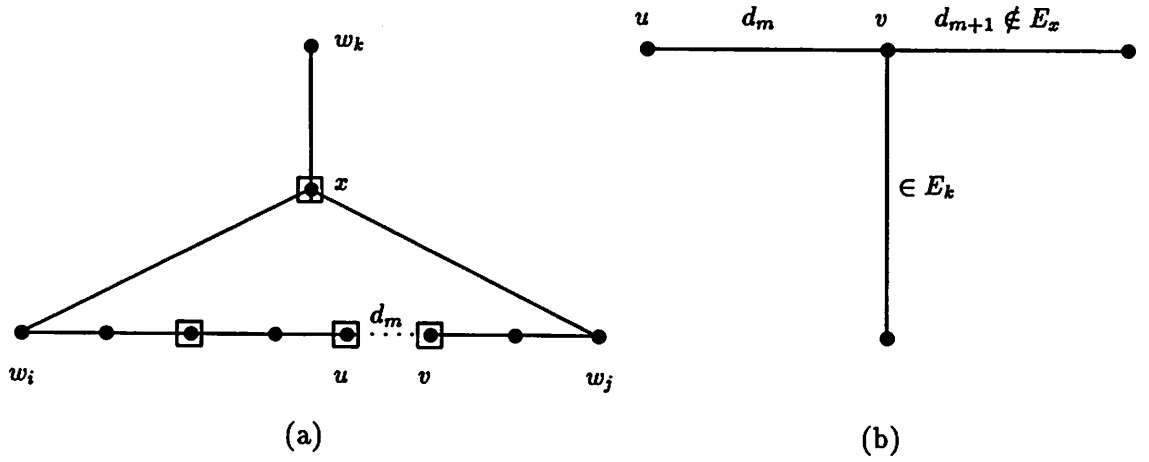


Figure 5: The distribution of edges in E_x on P_{ij}.

Now let $P_{12} = v_0, d_1, v_1, d_2, \ldots, d_r, v_r$ where $v_0 = w_1$ and $v_r = w_2$. For $i = 1, 2, 3$, let y_i be the last node of $P_{i,i+1}$ encountered when traversing $P_{i,i+2}$ from w_i to w_{i+2}, where addition is performed modulo 3. (See Figure 6.)

Claim 7 *The number of edges between w_i and y_i on $P_{i,i+1}$ is even.*

Proof. By symmetry it suffices to show this for the case of w_1, y_1 and P_{12}. From Claim 5, P_{13} cannot use any node on Q_{21} and P_{12} cannot use any node of Q_{31}, so $P_{13} \cap P_{12} \subseteq Q_{12} \cap Q_{13}$. According to Claim 4, the nodes of both Q_{12} and Q_{13} alternate between I_1 and $I_2 \cap I_3$. Therefore, if a node lies on both P_{12} and P_{13}, then its distance from w_1 along P_{12} must be of the same parity as along P_{13}. Suppose that y_1 is of odd (edge-) distance from w_1 on P_{12}; i.e., $y_1 = v_m$ with m odd. (See Figure 7.) Then y_1 is of odd distance from w_1 on P_{13} and, by Claim 6, the next edge on P_{13} must be in E_x. (Note that $v_m = w_2$ is impossible, since $w_2 \in Q_{21}$; thus d_{m+1} exists.) However, the only edge incident to v_m that is in E_x is d_{m+1} (since $d_m \notin E_x$ and $f \notin E_x$, where f is the edge of G that is incident to y_1 and is not on C_3). Therefore, v_{m+1} is the next node on P_{13}, which contradicts the choice of y_1 as the last node on P_{12} when following P_{13} from w_1 to w_3. □

Claim 8 *Let R_1 be the subpath of P_{12} from y_1 to y_2. Let R_2 be the subpath of P_{23} from y_2 to y_3 and let R_3 be the subpath of P_{31} from y_3 to y_1. Let C be formed by adjoining R_1, R_2 and R_3. Then C is an odd cycle.*

Proof. The paths R_1, R_2 and R_3 are disjoint by construction, except for their endnodes y_1, y_2 and y_3, so C is a simple cycle. Since y_1 is of even distance from w_1 on P_{12}, y_2 is of even distance from w_2 on P_{12}

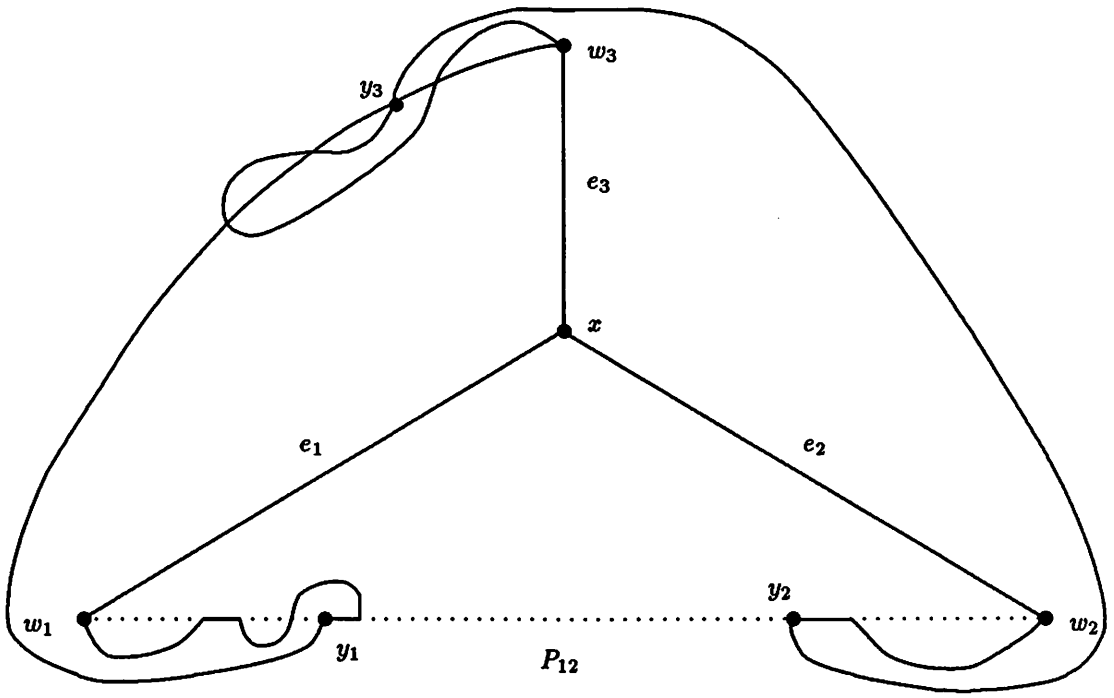


Figure 6: P_{ij}'s partitioned by the y_i's.

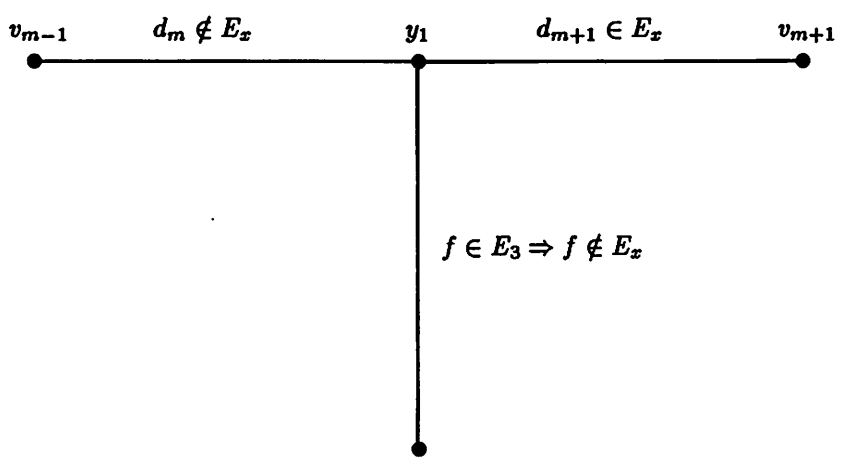


Figure 7: The status of the edges adjacent to y_1.

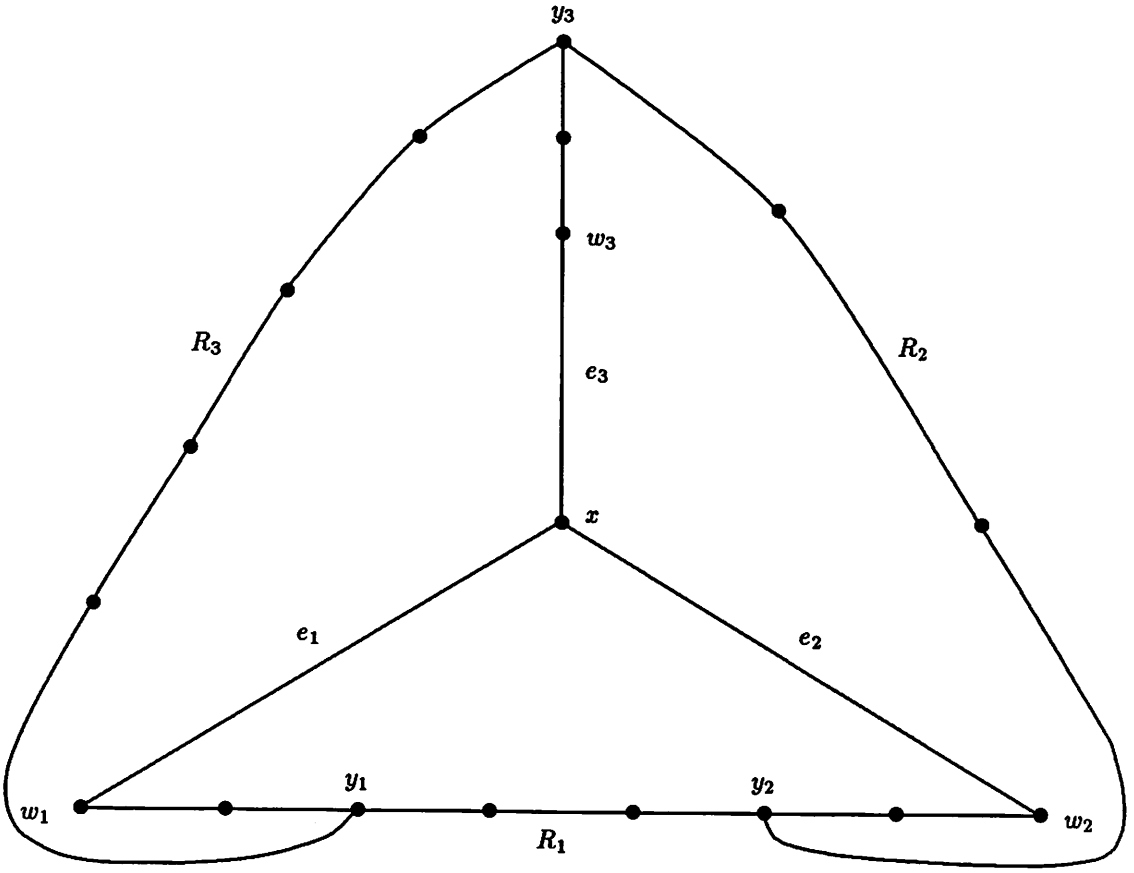


Figure 8: The subpaths R_1, R_2 and R_3.

and P_{12} is an odd path, then R_1 must be an odd path. The same holds for R_2 and R_3. Thus, C has an odd number of edges. □

Claim 9 *Let path S_1 be formed by adjoining (x, w_1) to the subpath of P_{12} from w_1 to y_1. Let S_2 be formed by adjoining (x, w_2) to the subpath of P_{23} from w_2 to y_2. Let S_3 be formed by adjoining (x, w_3) to the subpath of P_{31} from w_3 to y_3. Then S_1, S_2, S_3 and C form an even subdivision of K_4.*

Proof. Since y_i is of even distance from w_i on S_i, S_i must be an odd path, $i = 1, 2, 3$. By the choice of the y_i's, the S_i's and C are mutually disjoint, except for y_1, y_2 and y_3. From Claim 8, C is an odd cycle and y_1, y_2, y_3 divide it into arcs of odd length. □

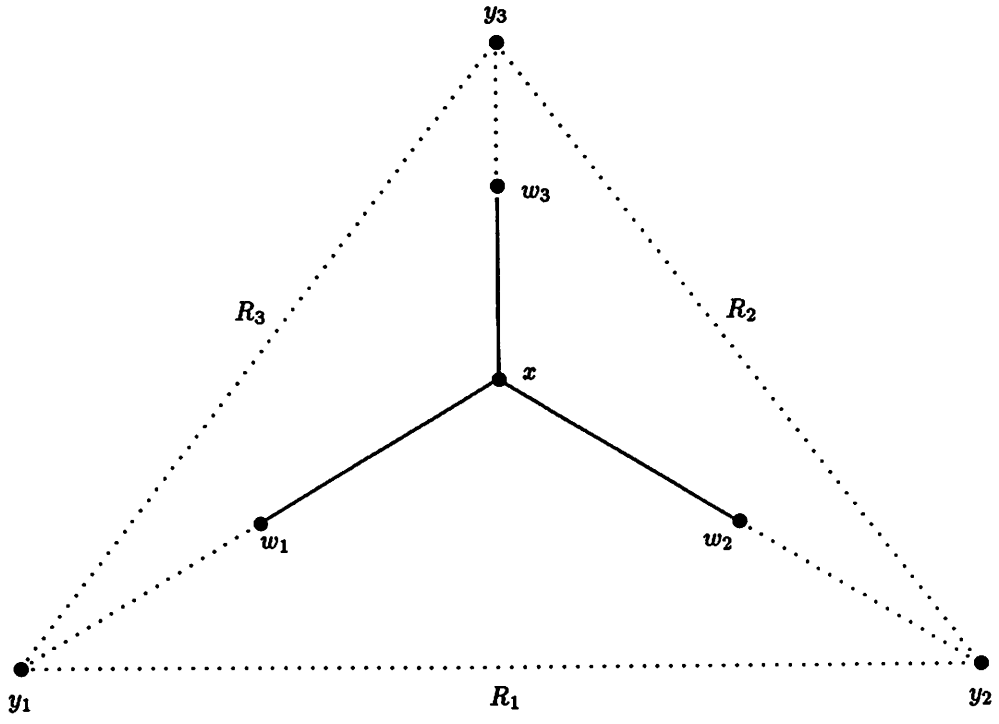


Figure 9: An even subdivision of K_4 centered at x.

The determination of the even subdivision of K_4 depicted in Figure 9 completes the proof of the theorem. □

References

[1] B. Andrásfai. On critical graphs. In P. Rosenstiehl, editor, *Theory of Graphs*, pages 9–19, New York, 1967. Gordan and Breach.

[2] V. Chvátal. On certain polytopes associated with graphs. *Journal of Combinatorial Theory, B*, 18:138–154, 1975.

[3] A. Hajnal. A theorem on k-saturated graphs. *Canadian Mathematics Journal*, 17:720–724, 1965.

[4] Frank Harary and Michael D. Plummer. On indecomposable graphs. *Canadian Journal of Mathematics*, 19:800–809, 1967.

[5] L. Lovász. Some finite basis theorems in graph theory. In A. Hajnal and V.T. Sós, editors, *Combinatorics II*, pages 717–729, Amsterdam, 1978. Colloq. Math. Soc. Janos Bolyai, 18, North-Holland.

[6] L. Lovász and M.D. Plummer. *Matching Theory.* Akademiai Kiado, Budapest and North-Holland, Amsterdam, 1986.

[7] L. Surányi. On line critical graphs. In A. Hajnal, R. Rado, and V.T. Sós, editors, *Infinite and Finite Sets (Colloq. Keszthely, Hungary, 1973), III*, pages 1411–1444, Amsterdam, 1975. Colloq. Math. Soc. Janos Bolyai, North-Holland.

INTEGER SOLUTION TO SYNTHESIS OF COMMUNICATION NETWORKS

S.Sridhar and R.Chandrasekaran. The University of Texas at Dallas, School of Management Box 830688, Gr 2.3, Richardson, Texas 75083-0688. U.S.A.

ABSTRACT: Given requirements between all pair of nodes, Gomory and Hu describe a polynomial algorithm to synthesize an undirected network with minimum sum of capacities of all arcs. A polynomial algorithm to produce integer solution to this problem is described in this paper. It is shown that rounding holds when there is no node with unit potential.

Introduction: The multiterminal network flow problem has been studied extensively in the literature. Apparently the first paper in this area is due to Kalaba and Juncosa [1]. A good description of the entire area is found in the book by Frank and Frisch [9]. There are three problems identified in these references: (i) analysis, (ii) realizability and (iii) synthesis. Analysis is concerned with the problem of determining the maximal flows between all pairs of nodes of a network (*nonsimultaneous - simultaneous flow goes under the name multicommodity flows*). Realizability is concerned with the problem of determining whether a given set of values is the set of maximum flow values of some network. Finally, the synthesis problem is concerned with designing a network that achieves specified requirements for all pairs of nodes at minimum cost.

There are many cases of each one of the above problem, depending on the assumptions regarding the network : directed [4,5] or undirected [6,7,8]; nature of costs - unit or arbitrary; nature of capacities - node [3] or arc; etc. In this paper, we discuss the synthesis problem on undirected networks with unit costs. This problem has been considered in many previous papers; for example, see the paper by Gomory and Hu [6]. In their work, they derive an elegant solution to the problem when every arc is allowed and capacities can be nonintegral. Indeed, their solution, in general, happens to be multiples of half-integers when the requirements are integral. In this paper, we present algorithms to produce integral optimal solutions

Problem: Let N be the set of nodes in a finite graph. Given non negative numbers $r_{x,y}$, representing the flow requirements between nodes x and y in N, the synthesis problem is to find arc capacities $c_{i,j}$ for i,j $\in$ N (we consider the undirected case; i.e.,

$c_{i,j}=c_{j,i}$) $\ni \sum_{i<j} c_{i,j}$ is minimum. In our case, we have an additional constraint that $c_{i,j}$ be integral, and hence, we may assume that the given requirements are all integral, and we do so from now on.

Definitions and Notation:

G^r : Requirement graph:
$G^r = [N, A^r]$ where $A^r = \{(x,y) : x,y \in N; r_{x,y} > o\}$

u_x $= \max_y \{r_{x,y}\}$ = potential of node x. *Please note that the maximum among all u's must be achieved by at least two nodes; in the course of the algorithm, this will be true except possibly at the very last step. Indeed , given set of positive numbers, they correspond to a set of u's iff there are at least two maximum values among the given set.*

N^i = Set of nodes at the i^{th} iteration.

n^i $= |N^i|$

v^i $= \min_{x \in N^i} \{u_x\}$

C : Integral circuit : An Hamiltonian cycle with node set N^i (the circuit must be in decreasing order of node potentials, in the clockwise direction) and capacities of each edge equal to $\left\lfloor \frac{v^i}{2} \right\rfloor$. (note $\left\lfloor \frac{v^i}{2} \right\rfloor = 0$ if $v^i = 1$, this will be defined as *zero integral circuit*)

D : Continuous circuit : An Hamiltonian cycle with node set N^i and arc capacities equal to $\frac{v^i}{2}$.

y^i = A node with the second lowest potential ($u_{y^i} > v^i$, for this definition), if more than one node has second lowest

potential then y^i is the node appearing at the latest position in the clockwise direction of circuit C.

M : Unit matching :*This is always done to a specified integral circuit C with a specified node numbering.* Nodes N^i of C are numbered from 1 to n^i as they appear in C (taken in the clockwise direction and [k] represents the node in the k th position in this numbering) with the additional requirement that y^i is $\left[\left\lceil\frac{n^i}{2}\right\rceil\right]$ if there is such a node. (Note : This numbering is independent of the ordering defined in C). The arcs A_M of M are $\left([i],[i+\left\lfloor\frac{n^i}{2}\right\rfloor]\right)$ with $1 \le i \le \left\lceil\frac{n^i}{2}\right\rceil$ and $c_{i,j}=1$ for all these arcs. *Note that the nodes of this matching are all in* N^i. *Actually, the term matching is an abuse of language; in the case when* n^i *is odd, one node has two arcs incident at it. Also, no matching problem is being solved.*

We wish to draw parallels to the algorithm of Gomory and Hu that solve the continuous version of the problem. One could view their algorithms as follows (*although it is not stated in this form*):

Algorithm GH:

Step 1: Compute node potentials;

Step 2: Let $i = 1$; $G^0 = [N, A^0 = \varnothing]$;

Step 3: $N^i = \{x : u_x > 0\}$; If $N^i = \varnothing$ stop;
G^{i-1} is optimal. Else go to step 4;

Step 4: Let $v^i = \min_{x \in N^i} \{u_x\}$;

Step 5: Let $G^i = [N, A^{i-1} \cup D]$;

Step 6: Let $u_x = u_x - v^i \; \forall \; x \in N^i$; $i = i + 1$; go to step 3.

A *phase* in this algorithm is step 3 - step 6. The capacities of arcs of the final solution is a *superposition* of their capacities of each

phase. The sum of the capacities of each phase equals $\frac{n^i \times v^i}{2}$, and this equals one half the reduction of node potentials in that phase. It is easy to show that the sum of all the capacities in the final network is at least equal to one half the sum of node potentials in the continuous case (see [2] for example). (Therefore sum of capacities, in the integral case, is at least equal to the smallest integer value higher than this quantity). During each of these phases, we send an additional v^i units of flow between any pair of nodes in the set N^i. Hence the algorithm works.

Now we will develop some preliminary results for the integer version of the problem, then present the algorithm and finally prove its validity.

Lemma 1: Let G^r be connected. If $u_x = 1 \; \forall \, x$ then any spanning tree of the graph with all arc capacities equal to 1 is the required integral solution.

Proof: Clearly, any optimum (solution) graph must be connected (this follows from the assumption that the requirement graph is connected) and none of the arc capacities need to be greater than one. If $\exists$ a circuit, then we can save by removing any arc of such a circuit. Hence the result.

Lemma 2: Let each component of the requirement graph have at least one node (actually in this case, it must be two) whose potential is greater than 1. Let $S = \{i : u_i = 1\}$ (note that $S \neq N$). Let c^0 be any feasible solution to the integer version of the problem. Then,

$$\sum_{i<j} c^0_{i,j} \geq \left\lceil \frac{\sum_{i \notin S} u_i}{2} \right\rceil + |S|.$$

Proof: Clearly, for any feasible solution to the continuous or integral version of the problem, $\sum_j c_{i,j} \geq u_i \; \forall \, i$ holds.

Let a set T be defined recursively as follows:

$T^0 = \{i \in S : \sum_j c^0_{i,j} = 1\}$; (all nodes with only one arc incident on it and capacity of that arc being one - this will be defined as *unit tip nodes*).

$T^k = T^{k-1} \cup \{i \in S : \sum_{j \notin T^{k-1}} c^0_{i,j} = 1\}$; (nodes which *became* unit tip node after removal of current unit tip nodes)

If $T^k = T^{k-1}$ define $T = T^k$ and stop.

Let $\hat{e}^0_{i,j} = \begin{cases} c^0_{i,j} & \text{if } i \text{ and } j \notin T \\ 0 & \text{otherwise} \end{cases}$

Hence,

$$\sum_{i<j} \hat{e}^0_{i,j} = \sum_{i<j} c^0_{i,j} - |T| \qquad \text{.....(1)}$$

Claim: $\sum_{i<j} \hat{e}^0_{i,j} \geq \left\lceil \frac{\sum_{i \notin S} u_i}{2} \right\rceil + |S - T|$ .....(2)

Proof: For c^0, since no node in T can be used as an intermediate node for satisfying flow requirement between any pair of nodes both of which are not in T (Because node in T^0 cannot be used as intermediate node $\Rightarrow$ nodes in T^1 cannot be used and so on up to T^k $\Rightarrow$ the above statement), $\hat{e}^0$ is a feasible solution to the integral version of the problem for the nodes in N - T with the original requirements between pairs of these nodes. Hence:

$$\sum_j \hat{e}^0_{i,j} \geq u_i \; \forall \, i \notin S;$$

$$\sum_j \hat{e}^0_{i,j} \geq 2 \; \forall \, i \in (S - T);$$

Now by adding these constraints and noting that the left hand side is integral, the claim follows.
The rest of the lemma follows by substituting (2) in (1).

Remark: By using similar argument, we can show that for any feasible integer solution we have: $\sum_{i<j} c^0_{i,j} \geq \left\lceil \left(\frac{\sum_{i \notin S} u_i}{2} \right) \right\rceil + |S| - k$, where k

is the number of components in G^r for which all node potentials are equal to 1.

The proof of the above lemma suggests the following. In any optimal solution, $\sum_j c_{i,j}$ is either 2 or 3 $\forall i \in S - T$. Moreover, at most, one such value is 3, and this happens only if $\sum_{i \notin S} u_i$ is odd. Also, there exists an optimal solution in which this value is 1 or 2 for all nodes in S. In any component of G^r for which some node potentials are greater than one, since changing these unit potentials to 2 does not affect the optimal value and yields a greater value for the flow between any such node and any other node, it is desirable to do this to get a "better" optimal solution - one that yields greater flow beyond the minimum required, without affecting the value of the objective function.

If $\sum_{i \notin S} u_i$ is odd, then we can increase the value of any u_i (except a node which is in a component of G^r, all of whose node potentials are equal to 1) by one without affecting the optimal value. If we increase u_i which is not the largest value, this will increase the flow that can be sent from that node to those nodes with a higher u. The choice of the node selected is arbitrary. Hence, in this case, we do not have a single maximum vector for the flow values unlike the continuous case. (see definition of $\bar{r}$ in [2]).

There is one more important difference between the continuous and the integral case. In the continuous case, the optimum value depends only on u_i and not on $r_{x,y}$. Thus, if G^r is not connected, this would not affect the value. This statement is true in the integral version too, if each component of the *requirement graph* has one or more nodes with $u_i \geq 2$ and $\sum_{i \notin S} u_i$ is even for that component. However, those components with all nodes having unit potential must be treated independently of the remaining requirements. It is desirable to treat all components of G^r which have some nodes with potential greater than 1 together; this is essentially important for

those components that have $\sum_{i \in S} u_i$ odd. If there are more than one such components, then we must at least pair wise group them in any optimal solution. This is not necessary in the continuous case.

Lemma 3: If no component of G^r has all node potentials equal to 1 (*note that in this case there are at least two nodes whose node potential is larger than one in each component of the requirement graph*), then $\exists$ an optimal integral solution in which $c_{x,y} = 1 \; \forall \; x \ni u_x = 1$ where y is some node with $u_y > 1$; $c_{x,z} = 0 \; \forall \; z \neq y$ in such an optimal solution.

Proof: If $u_x = 1$ for some (but not all) nodes, then $\sum_{y \in N} c_{x,y} \geq 1$ in any feasible solution. If $c_{x,y} = c_{x,z} = 1$ in an optimal solution (by lemma 2 $\exists$ a solution with $\sum_j c_{x,j} \leq 2$), then the solution with $c_{x,y} = 1$ and $\hat{c}_{y,z} = c_{y,z} + 1$ is also optimal. Thus, we can assume that for each node x with $u_x = 1$, there is exactly one arc $c_{x,y}$ with capacity equal to one and all other arcs capacities $c_{x,z}$ equal to zero. In addition, we may assume that if $c_{x,y} = 1$, then $u_y > 1$. Hence the lemma. *Please note that there is an optimal solution to the entire problem in which these nodes are not used as intermediate nodes for any flows between any pair of nodes neither of which has node potential equal to one. Better still we can increase these u_i to 2 without affecting the objective value; this will allow us to send more flows from these nodes to the remaining nodes.*

Lemma 4: Let G^i be a graph on the node set N^i whose arc set is defined as follows. If v^i is even, then G^i is C; if $v^i \geq 3$ and is odd, then G^i is union of C and matching M with respect to this circuit. (*Note that both* C *and* M *have well defined arc capacities*). Then G^i realizes flow v^i between all pairs of nodes in N^i.

Proof: If v^i is even, then the lemma follows from the fact that solution to the continuous problem is integral. When v^i is odd the lemma is trivially true for pairs $(x,y) \in A_M$. Let $v^i = 2w + 1$, where w is a positive integer and $(x,y) \notin A_M$. We want to show that

v^i units of flow can be sent from x to y. Let x = [j] and y = [k] with j and k positive integers. Without loss, we assume that $j < k < \left\lfloor \frac{n^i}{2} \right\rfloor$.

Let $s = j + \left\lfloor \frac{n^i}{2} \right\rfloor$; $t = k + \left\lfloor \frac{n^i}{2} \right\rfloor$ and $m = \left\lfloor \frac{n^i}{2} \right\rfloor$.

Send w units from x = [j] to [j+1] . . . [k] = y;

1 unit from x = [j] to [s], [s-1], . . . , [m], . . . ,[k] = y ;

w units from x = [j] , [j-1], [n^i], . . . ,[t];

1 unit from [t] to [k] = y;

w - 1 units from [t], [t-1], . . . ,[s], . . . ,[k] = y; achieving a total of v^i from x to y. (See figure 1). If n^i is odd, then there are two arcs of the matching incident at some node y; however, we will use only one of these two arcs to achieve this flow v^i; the other arc might be useful later on for some other purpose if G^{i+1} exists.

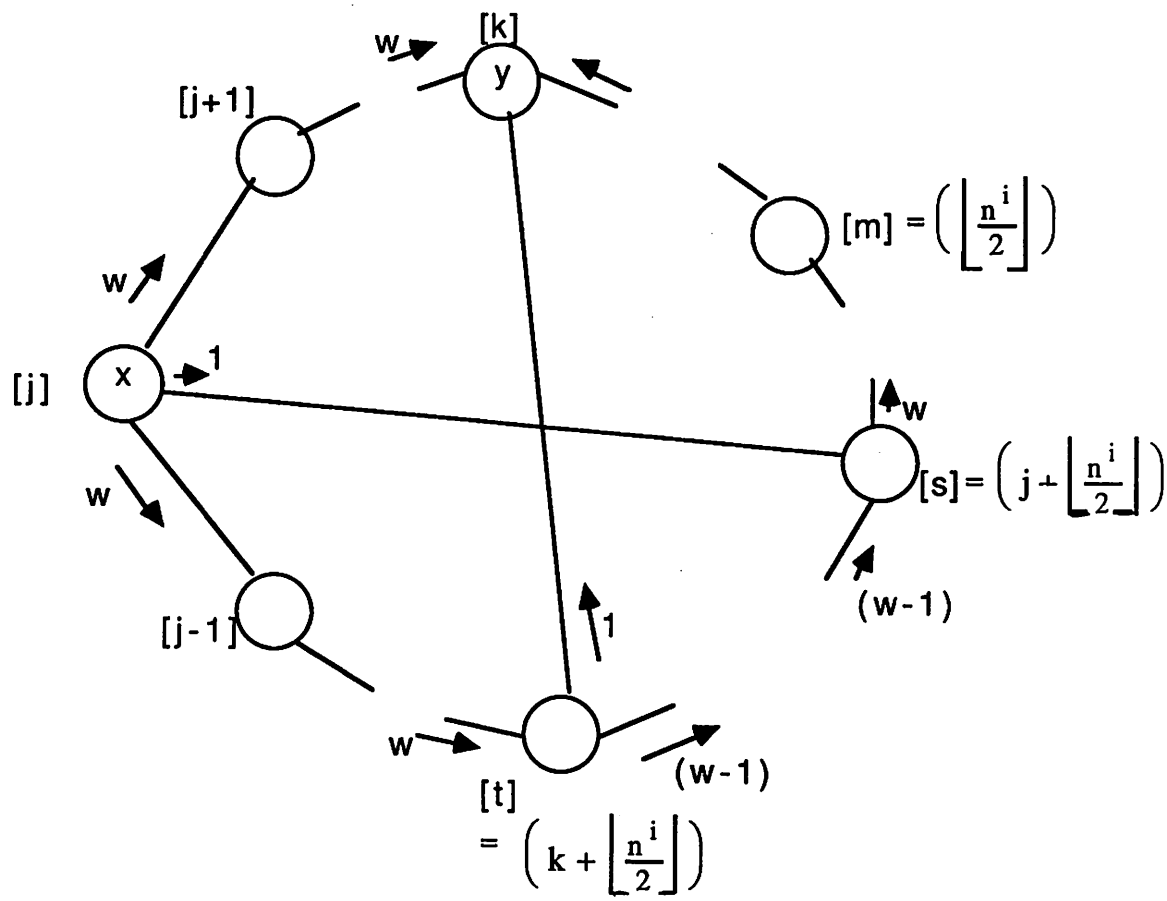


FIGURE 1

Now it is best to describe the entire algorithm in detail and then prove its validity.

Algorithm Synthesis:

Step 1: Compute node potentials $u_x \ \forall x \in N$

Step 2: If $u_x = 1$ for all $x \in N$ in any component of G^r, then any spanning tree of the component with unit capacities for all arcs is an optimal solution for this component. This set of nodes is deleted from further consideration.

Remark: Do *either* step 3 or step 3 (Alt.) *but <u>not</u> both.*.

Step 3: If $u_x = 1$ for some $x \in N$ (but there is no component of G^r with all node potential equal to one), then let $A = A \cup \{ (x,z) : x \ni u_x = 1 \}$ with $c_{x,y} = 1$, where z is some node $\ni u_z > 1$ (in fact there are at least two such nodes as we have seen earlier); Let $N = N - \{ x : u_x = 1 \}$.

Step 3(Alt.): Let $S = \{ x : u_x = 1 \}$ and suppose that there is no component of G^r with all node potentials equal to one, then let $u_x = u_x + 1 \ \forall x \in S$

Step 4: Let $i = 1$; $N^i = N$

Step 5: (a) Construct an integral circuit $G^i = C$

(b) If v^i is odd, then do a matching M with respect to C and let $G^i = C + M$.

(c) If v^i is odd and n^i is also odd, then let $u_{y^i} = u_{y^i} - 1$.

Step 6: $G = G + G^i$; $u_x = u_x - v^i \forall x \in N^i$;
$i = i + 1$; $N^i = N^{i-1} - \{ x : u_x = 0 \}$

Step 7: If $N^i = \varnothing$, then stop. If $n^i = 1$, then connect this node to a node with second highest node potential in N^{i-1} with capacity equal to the current potential of node in N^i and stop; Else go to step 5.

Theorem 1 : Algorithm synthesis is valid.

Proof : Complexity : Computation of node potential in step1 is of $O(n^2)$. Step 2 is of O(n). Steps 5 - 7 can at most occur n times; steps 5 and 6 are O(n) and step 7 takes constant time. Hence the complexity of the entire algorithm of $O(n^2)$.

Optimality and feasibility will be proved by lemma 5 and lemma 6, respectively.

Lemma 5: Algorithm synthesis is optimal.

Proof: If $u_x = 1 \; \forall \, x \in N$ the optimality (and feasibility) follows from lemma 1. Optimality (and feasibility) of step 3 follows from lemma 3. At the beginning of **algorithm synthesis**, nodes with potential one are take care of. At this stage, we know that there exists an optimal solution to the entire problem with

(i) one arc out of each node with potential one;

(ii) capacity of this arc is one;

(iii) this arc connects this node to a node with potential ≥ 2;

(iv) rest of the network is *optimal* for the original requirements between pairs of the remaining nodes.

Thus, from now on, we need to discuss only the case where no node potential is one. In such a case, in the **algorithm synthesis**, a *phase* corresponds to the interval between two non zero integral circuits. Within each phase, we have sub phases corresponding to the matching done to zero integral circuit.

At any time in the algorithm, half the value of the reduction in node potential exactly equals the increase in sum of the capacities, *except possibly at the very last step during which it could be precisely one half.* Hence the optimality of the algorithm.

Lemma 6: Algorithm synthesis produces a graph which realizes the required max flow between all pairs of nodes.

Proof : To show that requirements can be met, we show the actual flows for each pair. The flows of each phase can be superimposed (ie., added) but within a phase *we do have to reroute the flows as new matchings are added with zero full circuits.*

If v^i is even, then it is easy to verify that C will allow us to increase the flows between any pair of nodes in N^i by v^i units.

If v^i is odd and greater than one, then by lemma 4 it is easy to verify that C + M allows us to increase the flows between nodes in N^i by v^i units.

Hence superposition is valid as long as all G^i are of above two cases. The difficulty (of superposition) arises only when the following two instances occur.

Instance (a) : This concerns the reduction of node potential of y^i by *one more* when y^i is also odd. We want to call attention to the fact that, in this case, since y^i is the second least potential in N^i, there is at least one node h with $u_h \geq u_{y^i}$ in N^i (*except* possibly at the very last step which is handled in a different manner in step 7 of the algorithm). *What we need to show is that we can send the required flows between* y^i *and any other node* h *in* N^i *with* $u_h \geq u_{y^i}$.

We already know from lemma 4 that we can send v^i additional units between y^i and any such h using C and M. Since we reduced the node potential of y^i by $v^i + 1$ at this step, we know that *future* increases in capacity will take care of an amount of flow between these nodes equal to $(u_{y^i} - v^i - 1)$. *It is this last unit of flow that we are concerned about now.*

Instance (b) : When v^i is one, the matching is done on zero integral circuit. The G^i corresponding to this is *not* (by itself) capable of sending one unit flow between every pair of nodes, making rerouting necessary.

Note: 1. When n^i is odd instance (b) involves instance (a).

2. Occurrence of non zero integral circuit marks the end (beginning) of this (next) phase.

3. If G^i is C and G^{i+1} is matching on zero integral circuit no rerouting is necessary, by lemma 4. Whereas if G^i is (C + M) or matching on zero integral circuit and G^{i+1} is also matching on zero integral circuit then rerouting is

essential. Hence the occurrence of this instance implies the occurrence of two consecutive matchings.

Rest of the proof is to show feasibility during the occurrence of any of the above two instances.

For ease of the proof we define *internal path* as a simple path with the following properties.

(1) It has a definite starting point,

(2) Uses only matching arcs of successive sub phases (*zero full circuits*) (i^{th} and $i+1^{st}$ iteration).

(3) Uses arcs of i^{th} matching from higher potential nodes to lower potential nodes.

(4) Uses arcs of $i+1^{st}$ matching from lower potential nodes to higher potential nodes.

(5) Uses the i^{th} and $i+1^{st}$ matching arcs alternatively starting with $i+1^{st}$ matching arc.

Consider the following graphs,

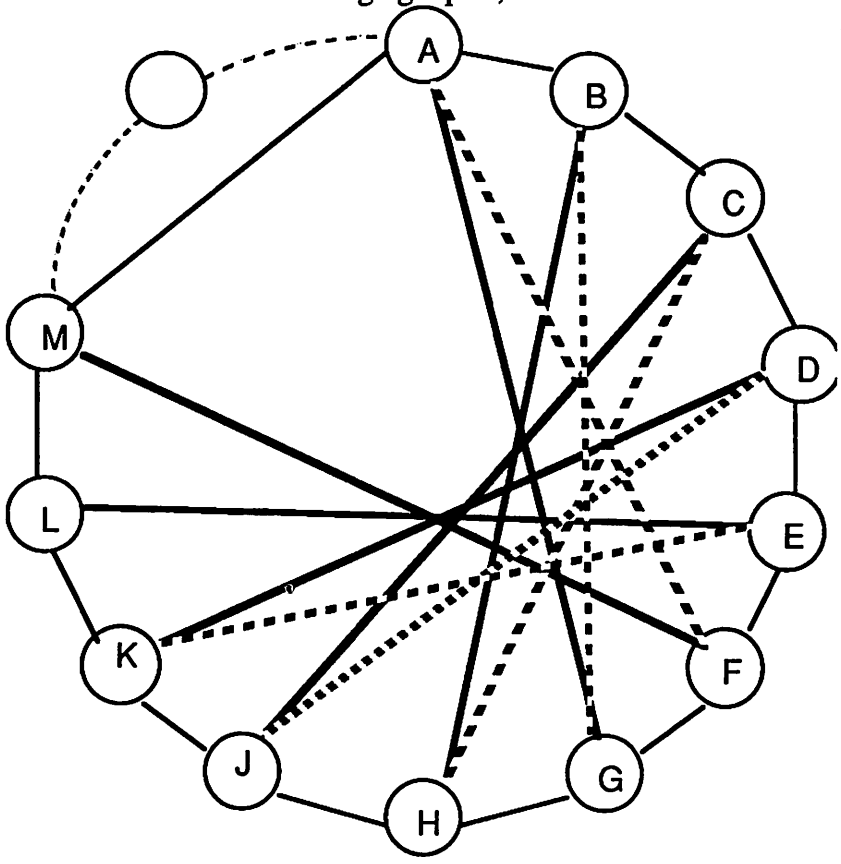


Figure 2

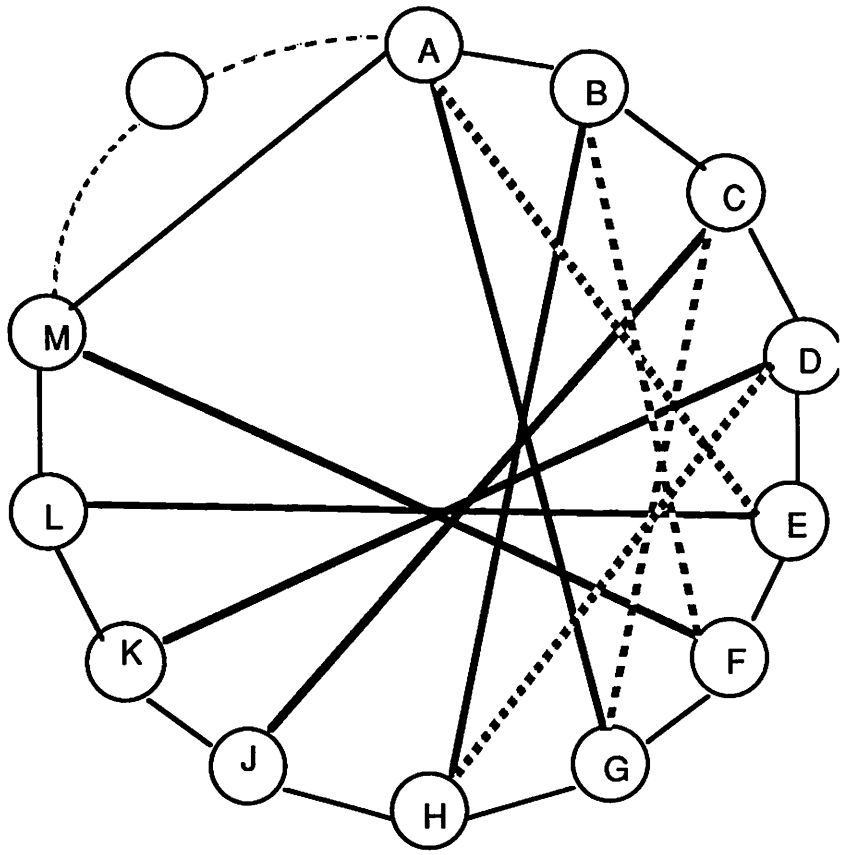


Figure 3

In figure 2 the internal path starting from G (*note, by definition the internal path is unique for a specified starting point*), is G-B-H-C-J-D-K-E, say P^0.

In figure 3, F-B-H-D is the internal path with the starting point at F, say P^1 and G-C is the internal path with the starting point at G, say P^2.

To prove the feasibility when instance b occurs: For clarity we will prove the feasibility in the above two examples and generalize them. In figure 2 , after the $i+1^{st}$ iteration we should be able to send one additional unit between all pairs of nodes in N^{i+1}. (A...K, in this example), specifically node A should have an augmenting path to any other node.

Clearly A-F is an augmenting path to F.

To G, the starting point of P^0, A-F$\xrightarrow{ccw}$B-G is an augmenting path (F$\xrightarrow{ccw}$ B : path from F to B in the counter clockwise direction on the circuit C, of the current phase), because path A$\xrightarrow{cw}$G will

have a flow w $(=\left\lfloor \frac{v^i}{2} \right\rfloor)$, enabling an augmenting of 1 unit from G $\xrightarrow{ccw}$ B.

To any other node X, from A, reach the staring point G with A-F $\xrightarrow{cw}$ G and terminate P^0 at X, since P^0 contains all other nodes. Eg.1, A-F $\xrightarrow{cw}$ G-P^0 terminated at C will be an augmenting path from A to C, since F $\xrightarrow{cw}$ G will have a flow of w-1 units in the *opposite* direction (note: if w = 1 then, there is no flow in F $\xrightarrow{cw}$ G and hence has excess capacity of 1 unit). Eg. 2, A-F $\xrightarrow{cw}$ G-P^0 terminated at J will be an augmenting path from A to J, since F $\xrightarrow{cw}$ G will have *positive* flow of w-1 units and hence has one excess capacity.

Thus A can send one additional unit to any other node in N^{i+1}. With the same argument B can send one additional unit to any other node except A and F. But, if there is an augmenting path from A to B there is an path from B to A. This is because max flow from A to B equals max flow from B to A, since the graph is undirected. Also if there is an augmenting path from B to A then there is a path from B to F, since A-F is an arc of the $i+1^{st}$ matching (note: if the augmenting path from B to A contains F then there is no necessity to go through A to reach F). With similar argument it is easy to see that there is an augmenting path of unit capacity between any two pair of nodes in set N^{i+1}.

In figure 3, there are two starting points (F and G) in order to find the augmenting path from A to any other node in N^{i+1}. (note: F and G are starting points with respect to A, since $(A,E) \in G^{i+1}$ and $(A,G) \in G^i$. In general, starting points w.r.t. X will be all the nodes between Y and Z, including Z and not including Y, in the clockwise direction of the latest non zero circuit, with $(X,Y) \in G^{i+1}$ and $(X,Z) \in G^i$). To reach any of the starting point, say G(F), use A-E $\xrightarrow{ccw}$ C(B)-G(F) as an augmenting path. Between any other pair, the argument is similar to the argument for figure 2.

Generalization: In general to send one unit from [s] to [t]

1. If $t = s + \left\lfloor \frac{n^{i+1}}{2} \right\rfloor$ then [s] - [t] is the necessary augmenting path.

2. If $s + \left\lfloor \frac{n^{i+1}}{2} \right\rfloor < t \leq s + \left\lfloor \frac{n^{i}}{2} \right\rfloor$ then [t] is a starting point w.r.t [s], hence [s] - [$s + \left\lfloor \frac{n^{i+1}}{2} \right\rfloor$] $\xrightarrow{ccw}$ [$t - \left\lfloor \frac{n^{i+1}}{2} \right\rfloor$] - [t] is the necessary augmenting path.

3. For any other node, [s] - [$s + \left\lfloor \frac{n^{i+1}}{2} \right\rfloor$] - (*relevant* starting point) - (internal path terminated at [t]) is the necessary augmenting path.

The starting point corresponding to [t] can be found by backtracking from [t], following the definition of internal path, from the opposite direction.

The only remaining part is to show that the internal path allows flow augmenting. During the flow augmenting the $i+1^{st}$ matching arcs are used to send one unit of flow from lower potential nodes to higher potential nodes *or* not used at all (depending on the starting point of the internal path, if there are more than one starting point). If iteration i + 2 is also a matching on zero integral circuit then the flow augmenting will use the $i+1^{st}$ matching arcs from higher potential nodes to lower potential nodes, therefore these arcs either have one unit of flow in the opposite direction or do not have flow at all, in either case they serve in the internal path as an augmenting path. Hence the result.

To prove the feasibility when instance a occurs: Two cases must be discussed in this section.

Case 1: When y^i is not an element of N^{i+1} : At the i^{th} step y^i is matched to two nodes , say x and y (with potential greater than y^i, by definition of C) and as seen in lemma 4, only one of the two (say using (y^i,x)) matching arcs will be used to send 2w+1 units of flow. To send one additional unit of flow from y^i to all the nodes with higher potential, send one unit to node y. Since G^{i+1} is capable (even if it is a matching on zero integral circuit, as shown in the proof of the previous section) of sending at least one more unit between all nodes in N^{i+1} , y can send this additional unit to any other node in N^{i+1}.

Case 2: when y^i is an element of N^{i+1}

$\Rightarrow$ y^i is the *unique* node with the minimum node potential

$\Rightarrow$ either (a) this will be the only node to drop out of N^{i+1}
or (b) this will be one of the two nodes dropping out of N^{i+1} (if N^{i+1} is also odd and $u_{y^{i+1}} = 2$, also note $v^{i+1} = u_{y^i} = 1$).

(a) $\Rightarrow$ G^{i+2} involves all nodes of N^{i+1} except y^i. Hence, with the same definition of x and y as in case 1, the additional one unit can be sent to y, and since y is an element of N^{i+2}, it is capable of sending at least one unit between all other nodes in N^{i+2} using G^{i+2}, the result follows.

(b) $\Rightarrow$ y^i is matched with two nodes s and t in G^i and neither s nor t is y^{i+1}. Also, y^{i+1} is matched with two nodes p and q in G^{i+1} (p and q could be same or different from s or t). we must prove that y^i can send u_{y^i} units to y^{i+1}.

y^i can send $u_{y^i} - 2$ units using graphs $G^1 \ldots G^i$, and either using (y^i,s) or (y^i,t) but not both, (say (y^i,s)). It can send one more unit due to G^{i+1}, using either (y^{i+1},p) or (y^{i+1},q) but not both, (say (y^{i+1},p)). Hence, the additional one unit can be sent using the path y^i - t - . . . - y^{i+1}. Path t . . . q exists because p,q,s and t all appears in G^{i+1} and hence capable of sending at least one unit among each other, (note that if G^{i+1} does not exists then this case is trivial due to step 7 of the algorithm).

This completes the proof of lemma 6 and consequently proving the validity of the algorithm.

REFERENCES:

[1] R.E. Kalaba & M.L. Juncosa, (1956), *Optimal Design and Utilization of Communication Networks*, Man. Sci., 3, pp. 33-44.

[2] L.R. Ford, Jr. & D.R. Fulkerson, (1962), *Flows in Networks*, Princeton Press, NJ.

[3] F. Granot & R. Hassin, (1986), *Multiterminal Maximum Flows in Node-Capacitated Networks*, Discrete Appl. Math., 13, pp. 157-163.

[4] D. Tang & R.T. Chien, (1961), *Analysis and Synthesis Techniques of Oriented Communication Nets*, IRE Trans. on Circuit Theory CT-8, pp. 39-44.

[5] O. Wing & R.T. Chien, (1961), *Optimal Synthesis of a Communication Net, IRE Trans. on Circuit Theory* CT-8, pp. 44-49.

[6] R.E. Gomory & T.C.Hu, (1961), *Multiterminal Network Flows*, J. SIAM, 9, pp. 551-570.

[7] R.T. Chien, (1960), *Synthesis of a Communication Net*, IBM J. Res. Develop, 3, pp. 311-320.

[8] W. Mayeda, (1960), *Terminal and Branch Capacity Matrices of a Communication Net*, IRE Transaction on Circuit Theory CT-7, pp. 261-269.

[9] H. Frank & I.T. Frisch, (1971), *Communication, Transmission, and Transportation Networks*, Addison-Wesley, Reading, MA.

SCHEDULING MULTIPLE VARIABLE–SPEED MACHINES

Michael A. Trick *

April, 1990

Abstract

We examine scheduling problems where we control not only the assignment of jobs to machines, but also the time used by the job on the machine. For instance, many tooling machines allow control of the speed at which a job is run. Increasing the speed incurs costs due to machine wear but also increases throughput. We discuss some fundamental scheduling problems in this environment and give algorithms for some interesting cases. Some cases are inherently difficult so for these we give heuristics. Our approach illustrates the exploitation of underlying network structure in combinatorial optimization problems. We create heuristics that optimally schedule a large portion of the jobs and then attempt to fit in the remainder. This also gives a method for quickly finding valid inequalities violated by the linear relaxation solution. Our heuristics are compared to other, classical, heuristics.

Traditional sequencing and scheduling models assume that the time a job requires on a machine is not under control: either the time is fixed, or it is a random variable determined by outside forces. Some problems require not only the assignment of job to machine but also the choice of processing time, reflecting physical capabilities of the machines or extended possibilities in the model. In this paper, we examine some optimally solved special cases and some heuristics in this environment, which we call variable–speed scheduling.

One example of variable–speed machines occurs in the scheduling of tooling machines. Tooling machines take pieces of wood, plastic, or metal and, through cutting and planing, make smaller pieces of the desired shapes and sizes. In such applications, tool wear is considerable. It is possible to decrease tool wear by running the machines at lower speeds (shallower cuts, more planing passes) but this increases the time spent by the piece on the machine. Another example occurs in the assignment of insurance claims to investigators. More time spent by the investigator generally means reduced payouts (the investigator finds more cases of fraud and the claimant is more willing to settle to avoid protracted investigation) but each investigator has only a limited amount of time.

Many articles have been written developing methods for optimally cutting one piece on one machine (for a survey, see [7], with other papers being [14, 9]) but relatively little work has been done on handling multiple jobs. The exceptions are [2, 19, 20] who develop algorithms for precedence constrained single machine scheduling with a variety of optimality conditions. We examine a model that is both more difficult, for we allow multiple machines, and simpler, for we do not permit precedence constraints.

In addition to solving an interesting and useful problem, a goal of this paper is to illustrate the exploitation of embedded network structure in combinatorial optimization. We choose a formulation of this problem that has a generalized network problem as a linear relaxation. We can then find optimal solutions for the relaxation very quickly. More importantly, the structure of the basis of the relaxations contains an enormous amount of information. We can optimally schedule a large portion of the problem, leaving just a few jobs to be rescheduled. Furthermore, we can use the relaxation to heuristically reschedule the remainder. We also find violated inequalities directly from the basis structure.

The general model is as follows. We have n jobs (indexed by j) and m machines (indexed by i). The machines are not assumed to be identical, or otherwise similar in characteristics. Each job must be assigned to the machines. If job j is assigned to machine i, the minimum amount of time the machine will take is l_{ij} and a cost of c_{ij} is incurred. The job can take up to u_{ij} units of time on the machine. For each unit beyond the minimum, a profit of s_{ij} is received. Machine i has b_i units of time available. The objective is to minimize the total cost.

*Address: Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, USA, 15213.

We look at a number of variations on the basic model. The fundamental distinction is whether a job can be processed by more than one machine (divisible jobs) or it each must be assigned to exactly one machine. For models with divisible jobs, the size and cost of the assignment is prorated by the fraction of the job on the machine. Other variations include two–speed machines, where each job must be run at either its fast speed or its slow speed (each job takes either its maximum time or its minimum time), and restrictions on how speeds can change (by job, by machine, or by job and machine).

We begin by giving algorithms for finding the optimal solution for two very simple cases: one machine and multiple machine problems with divisible jobs. Although the algorithms are straightforward (the first is the greedy algorithm, the second is a generalized network). the solution structure is used as a building block for more difficult cases.

We then examine is the multiple machine, indivisible job case. The case where there is no flexibility in the machine speeds (i.e. $l_{ij} = u_{ij}$ for all i and j) is the generalized assignment problem. Because this problem is HP–hard, the variable–speed problem is as well. In [17], we examine the linear relaxation of the generalized assignment problem and find much structure. This leads to a heuristic with good behavior, both theoretically and practically. This also gives a method for generating violated inequalities, which improves the lower bound and decreases the gap between the lower bound and the heuristic solution. This paper generalizes these results for the more general case and presents a method for further decreasing the duality gap.

We conclude with computational results, comparing our heuristics with standard greedy and savings–regret heuristics for this problem. Our heuristics are robust, accurate, and fast.

1 Single Machine

We begin with the single machine case. A formulation of this problem is (in this section we repress the machine subscript i):

Minimize $\sum_j (c_j - s_j(x_j - l_j))$
Subject to
$\sum_j x_j \leq b$
$l_j \leq x_j \leq u_j$ for all j.

The x_j are interpreted as the amount of time spent by job j on the machine.

This problem is simply a linear knapsack problem with lower and upper bounds. There is a very simple solution procedure. First, determine if the instance is feasible by checking that the sum of the minimum times on the machine is no more than the time available on the machine. If that is the case, we can assign every x_j its minimum value. Our objective is to assign the remaining time of the machine so as to maximize our profit. The solution procedure is well known: sort the variables by decreasing s_j. In this order, increase the variables until the variable reaches its upper bound or the knapsack is filled. In the former case, continue with the next variable; in the latter, the remaining variables stay at their lower bound.

There are three sets of variables in the solution (any of which may be empty): A set U of variables at their upper bound, a set L with values at their lower bound, and a single variable j^* with a value in between. Furthermore, $s_j \geq s_{j^*} \geq s_k$ for any $j \in U$ and $k \in L$. In this model, the interpretation is that every job except one is processed using either its minimum time or its maximum time, and those that use their minimum times are those that do the least harm to the machine.

2 Multiple Machines, Divisible Jobs

For the multiple machine case, there are m machines with differing characteristics. One formulation for this problem is:

(LP1)

Minimize $\sum_i \sum_j (c_{ij} + s_{ij} l_{ij}) z_{ij} - \sum_i \sum_j s_{ij}(x_{ij} - l_{ij})$

Subject to

$\sum_j x_{ij} \leq b_i$ for all i,

$\sum_i z_{ij} = 1$ for all j

$l_{ij} z_{ij} \leq x_{ij} \leq u_{ij} z_{ij}$ for all i, j,

$x_{ij} \geq 0, z_{ij} \geq 0$ for all i, j.

(The objective above differs from the true value by the constant $\sum_i \sum_j s_{ij} l_{ij}$.) The z_{ij} are interpreted as the fraction of job j assigned to machine i and x_{ij} is time job j spends on machine i. Practically, the requirement that a job be completely scheduled makes feasibility a real issue. Normally a slack variable with large cost is added to each constraint. If the slack variable is used then the job is not completely scheduled.

This formulation is sufficient to show the polynomiality of this problem: simply solve the above model with a polynomial time linear programming algorithm. There is an alternative formulation, however, that models the problem as a generalized network. This formulation not only decreases the time required to find a solution, but also gives us insight into the structure of the optimal solution. This insight is missing in LP1.

Consider the linear program (with variables y_{ij} and y'_{ij}):

(LP2)

Minimize $\sum_i \sum_j c_{ij} y_{ij} - \sum_i \sum_j (c_{ij} - s_{ij}(u_{ij} - l_{ij})) y'_{ij}$

Subject to

$\sum_j (l_{ij} y_{ij} + u_{ij} y'_{ij}) \leq b_i$ for all i,

$\sum_i (y_{ij} + y'_{ij}) = 1$ for all j,

$y_{ij}, y'_{ij} \geq 0$ for all i, j.

The y_{ij} are interpreted to be the fraction of job j using the minimum time on machine j and y'_{ij} the fraction using the maximum time. Since a job using an intermediate time can be thought of as some portion using the maximum time and the rest the minimum time, the equivalence of LP1 and LP2 is obvious.

We get far more information out of the solution to LP2 than is obvious from the solution of LP1. Because each variable appears in just two constraints (plus non-negativity), LP2 is a generalized network problem. Therefore, an optimal solution has a lot of structure. In particular, the arcs that form thee basis form a set of one-trees (trees with one extra edge). From restrictions on the number and form of basis cycles (see [1]), it is easy to prove the following about the structure of any basic solution:

Theorem 1 *Any basic solution to LP2, represented graphically, is a set of connected components, where each component has exactly one of:*

1. *a job not completely scheduled;*
2. *a machine not used to capacity;*
3. *a job using an time between its maximum and minimum; or*
4. *a cycle of alternating jobs and machines.*

PROOF: This follows from the characterization of [1] and the recognition that for a job to run at a time between its maximum and minimum requires both y_{ij} and y'_{ij} in the basis, forming a basis cycle. □

We call a job using a time between its maximum and minimum time an *intermediate job*. We call a job that is partially assigned to more than one machine a *split job*. Based on this theorem, there are a number of easy corollaries.

Corollary 1 *There is at most one intermediate job on any machine.*

Corollary 2 *For every job, it uses an intermediate time on at most one machine.*

PROOF: These corollaries follow from condition 3. □

Corollary 3 *If a machine has an intermediate job assigned to it, then the machine is used to capacity.*

PROOF: If not, then both conditions 1 and 3 would hold for the component. □

Corollary 4 *The sum of the number of split jobs and the number of intermediate jobs is no more than the number of machines scheduled to capacity.*

PROOF: For each split or intermediate job, we will associate a machine scheduled to capacity. Each machine scheduled to capacity will be associated with at most one job, so the corollary follows.

Take a split job not on a basis cycle. Since it has degree at least two in the basis, associate with it any machine node one further from the basis cycle. For any split job on a basis cycle, orient the cycle arbitrarily (but consistently for every node on the cycle). Associate with each split job the next machine node on the cycle. Finally, for any intermediate node, associate it with the machine on which it uses an intermediate size. Every split and intermediate job is now associated with a machine; no machine is associated with more than one job; no machine not used to capacity is associated with any job. □

This structure will be used for heuristics in the case that the jobs are indivisible.

3 Multiple Machines, Indivisible Jobs

In many models, a job represents a discrete unit. The goal is then to assign jobs to machines so that each job is assigned to exactly one machine, minimizing cost. Again, we can choose the speed at which to do a job. We get integer programming formulations by adding constraints $z_{ij} \in \{0,1\}$ to LP1 and $y_{ij} + y'_{ij} \in \{0,1\}$ to LP2. Call the resulting problems IP1 and IP2.

In general, this problem is difficult (if $l_{ij} = u_{ij}$ for all i and j then the problem is the NP–hard generalized assignment problem [4]). Therefore, we address the problem of finding a good heuristic.

One approach is to use the results from the previous section as a starting point. We know that if there are a large number of jobs and a small number of machines then almost all jobs will be assigned to one machine each. By corollary 4, at most m jobs will be split jobs. This leaves us just a small number of jobs to schedule.

One approach to this rescheduling is to use the linear relaxation on the remainder. It is important to ensure that we do not cycle, generating the same solution over and over. We can do this by modifying the instance in a way to keep the integer program the same, but changing the linear program.

Consider a job j and machine i. If $u_{ij} > b_i$, then clearly we can reduce the upper bound to b_i without changing the integer program. Furthermore, if the resulting upper bound is less than the corresponding lower bound, then the variable can be deleted altogether. This process does change the linear relaxation, however. We call this process *reduction*.

Theorem 2 *If all variables for jobs uniquely assigned to a machine are fixed, at least one variable can be reduced relative to the resulting problem.*

PROOF: Consider the placement of all split jobs. There are two possibilities: either there exists a split job not on a basis cycle, or all split jobs are on the basis cycles.

Suppose there is at least one job not on the basis cycle. Consider a split job j that is maximally distant (when measured in number of edges) from the basis cycle for its component. Since it is split, there exists at least one machine node i distance 1 further from the cycle. This machine node is otherwise incident to uniquely assigned jobs. Therefore, once the uniquely assigned jobs are fixed, the remaining capacity of the machine is exactly the flow on arc (i,j) times the size associated with (i,j). The first value is less than 1, since j is split. The second is at most the maximum size of j on i. Therefore the resulting machine capacity is less than the maximum size of j on i.

Suppose all split jobs are on the basis cycles. Consider a machine i node on a cycle. Let its incident job nodes be j and k and suppose the size corresponding to the basic variable between j and i is no more than that between k and i. If the incident flow values sum to less than 1, then the new capacity of i will be strictly less than the maximum size of k (since it is a combination of two numbers no more than the maximum size

of k, and the weights sum to less than 1). If the incident flow is more than 1, then there exists a node on the cycle with incident flow less than 1. Finally, if the incident flow equals 1 for all machine nodes around a cycle, then either there exists a machine node where the size of j is strictly less than the size of k or j and k have the same size for all machine nodes. In the former case, the new capacity of the machine is less than the maximum size of k. In the latter, the basis is degenerate. □

This gives us a heuristic for this problem: fix all the values for jobs that are not split, reduce the variables, and resolve the linear relaxation. More formally, the heuristic is:

LR–Heuristic (Variable–speed)
Input: Variable–speed problem, with n, m, b_i, l_{ij}, u_{ij}, c_{ij}, and s_{ij} as defined in the text.
Output: Heuristic solution y, y'.

Step 0) Reduce all variables. If no variable remains, then go to Step 3.

Step 1) Solve LP2 to get solution y, y'.

Step 2) For each job assigned to a unique machine, fix the corresponding variables, delete the job from the problem, and update the machine capacities. Go to step 0.

Step 3) Given the machine assignments, optimally schedule the individual machines using the single machine algorithm.

Theorem 3 *Step 1 is executed at most $m+1$ times.*

PROOF: We know from the previous theorem that a new variable gets reduced each iteration. In order for a variable to get reduced, it was necessary that at least one job be scheduled. After the first iteration, only m jobs remain to be scheduled. □

This approach has many advantages over other heuristics. First of all, the initial generalized network gives a lower bound on the solution, so there are bounds on the deviation from optimality. Second, the relaxation can be solved very effectively with the generalized network simplex method ([1, 13]), so solution can be found very quickly.

One disadvantage of this approach is that theorem 2 requires fixing the size as well as the assignment of jobs. It would be better to only fix the assignment (the integer variable) and leave free the size (the continuous variable). It is not possible to ensure a reduction in this case, however. It is possible to check for what might be called a *strong reduction*: a reduction that occurs even if all variables take on their minimum size. In cases when a strong reduction prevents cycling, it is better to do only the strong reduction, fixing the assignments but not the sizes. Only when no strong reduction is available is the regular reduction done.

It is not possible to bound the deviation from optimality for this heuristic for a general cost function (nor could we for any heuristic unless $P = NP$). However, either a dummy machine with high costs has been added or slack variables with high cost are added, we can bound the number of unscheduled jobs. From Corollary 4 it is clear that the difference between the optimal number scheduled and the number the heuristic schedules is no more than m. We can reduce this slightly.

Theorem 4 *For any cost that maximizes the number of jobs scheduled, LR–Heuristic schedules no more than $m-1$ fewer than optimal.*

PROOF: It is possible to show that one variable doesn't become reduced, so at least one variable can be scheduled in the second iteration. See [17] for details. □

4 Violated Valid Inequalities

In the previous section, we showed how to use the linear relaxation solution to find good, heuristic, solutions. In this section, we show how to use the structure of the basis to improve the lower bound. In particular, we will show how to generate a violated inequality directly from the relaxation solution.

The formulation in IP1 is a fixed cost network flow problem with lower and upper bounds. Van Roy and Wolsey [18] have examined such problems and have devised some valid inequalities. We show that one

of the classes they present must have a member that is violated by the linear relaxation. Furthermore, the violated member is easy to find. This gives us the first step in a cutting plane algorithm: we can solve the relaxation and add a first set of cuts. After adding those cuts, however, we cannot iterate, for the resulting problem does not have a generalized network basis structure.

Consider a machine i and a subset J of jobs, partitioned into J_1 and J_2 (either may be empty) such that

$$\sum_{j \in J_1} l_{ij} + \sum_{j \in J_2} u_{ij} = b_i + \lambda$$

with $\lambda > 0$. Such a pair (j_1, j_2) is a special sort of *generalized flow cover*. The following is a special case of the constraints presented in [18]. Let $(a)^+$ be a if $a \geq 0$ and 0 otherwise.

Theorem 5 *If (J_1, J_2) is a generalized flow cover for machine i, then the inequality*

$$\sum_{j \in J_2} (x_{ij} + (u_{ij} - \lambda)^+ (1 - z_{ij})) + \sum_{j \in J_1} ((l_{ij} - \lambda)^+ + \min(l_{ij}, \lambda) z_{ij}) \leq b_i$$

is a valid inequality for the variable-speed scheduling problem.

PROOF: This is simply Corollary 3 of [18] with $C_2 = \emptyset$. □

What makes this special case particularly interesting is that there is a constraint of this class for every reduced assignment found in Theorem 2.

Theorem 6 *If the linear relaxation is solved, then there exists a violated inequality from the class of inequalities in Theorem 5.*

PROOF: Consider a solution to LP1, with x_{ij} representing flow and the z_{ij} denoting the fraction of assignment (i, j) used. Note that we can generate this solution from the solution to LP2 by setting $z_{ij} = y_{ij} + y'_{ij}$ and $x_{ij} = l_{ij} y_{ij} + u_{ij} y'_{ij}$.

Every variable (i, j^*) to be reduced has the following structure: If you take the jobs uniquely assigned to the machine (all at either their minimum size or maximum size) plus either l_{ij^*} or u_{ij^*} (depending on which is basic) then the total size is more than b_i. Therefore, if you take J_1 to be all variables at their lower bounds (including j^* if y_{ij^*} is basic) and J_2 be those at their upper bounds (including j^* if y'_{ij^*} is basic), then (J_1, J_2) is a generalized flow cover.

Now, examine the valid inequality in Theorem 5 with respect to the linear relaxation solution. Suppose $j^* \in J_1$. Since for all $j \in J_1/j^* \cup J_2$, $z_{ij} = 1$, the left hand side reduces to:

$$\sum_{j \in J_2} x_{ij} + \sum_{j \in J_1/j^*} l_{ij} + (l_{ij^*} - \lambda)^+ + \min(\lambda, l_{ij^*} z_{ij^*}).$$

Clearly $l_{ij^*} > \lambda$ since all of $J_1 \cup J_2/j^*$ fits on machine i. Therefore, the sum becomes:

$$\sum_{j \in J_2} x_{ij} + \sum_{j \in J_1/j^*} l_{ij} + (l_{ij^*} - \lambda) + l_{ij^*} z_{ij^*}$$

But the above sum without the term $(l_{ij^*} - \lambda)$ equals b_i, so the total sum is greater than b_i. Therefore the linear relaxation violates this valid constraint.

The case where $j^* \in J_2$ is similar. □

Therefore, we can quickly find a valid inequality that the linear relaxation solution violates. We can then add this constraint to LP2 (doing the necessary variable substitutions) and resolve. Normally, this will improve our lower bound.

The above theorem corresponds to regular reduction. There is a stronger constraint that corresponds to strong reduction when the corresponding variable is deleted (since the upper bound is less than the lower bound). In this case, we have identified a set of variables S such that even if all variables are at their lower bound, the total size is too large for the machine. Therefore we can generate the constraint

$$\sum_{j \in S} z_{ij} \leq |S| - 1$$

which will be violated and, in general, will be stronger than the constraints in theorem 5. As stated in the previous section, however, strong reduction is not guaranteed.

These constraints are not enough to create a cutting plane algorithm. Once we add the constraints, the generalized network structure of the problem is lost, so we can not repeat the process. It may be that the first round of cuts is sufficient to strengthen the lower bound, particularly if the cuts are lifted or otherwise strengthened (see [12] for a survey on strengthening cuts).

5 Other Heuristics

Solving the linear relaxation is only one heuristic for the multiple machine, indivisible job, problem. In this section, we will outline two other heuristics that form the base of our computational tests.

Because we know an optimal method for single machines, we can simply search for assignments of jobs to machines. One we have an assignment of jobs to machines, we can apply our single machine algorithm to determine the optimal sizes. The difficulty is in the assignment of jobs to machines.

The first heuristic is a greedy heuristic. Assume we have an arbitrary ordering of the jobs (here denoted $1, 2, \ldots, n$, but in general a permutation of this set). We divide the problem into n stages, where the subproblem to solved at stage j is the assignment of the jth job, given the assignment of jobs 1 through $j-1$.

We solve the subproblem in the most straightforward way: during stage j, we determine the cost of assigning j to each machine i in turn. We assume the assignments (though not the sizes) of 1 up to $j-1$ are fixed, and no job after j is considered. The job is then assigned to the machine that gives the minimum total cost.

Clearly, this is a very simple-minded heuristic. Jobs that appear early in the sequence ignore those that appear later, so assignments are made as though the machine is not filled to capacity. Jobs that appear late in the sequence might have to use very expensive machines in order to satisfy feasibility. One advantage of this heuristic, however, is that different orderings of the jobs lead to different solutions. This heuristic is good for generating many different solutions.

Our second heuristic attempts to handle the problem of jobs being left with only one expensive machine. Here there is no ordering of jobs. The jobs are assigned to machines one at a time. The job to be assigned is chosen by determining the maximum *regret* for not being able to use its best machine. The regret is defined to be the difference in costs between assigning it to its best machine and assigning it to its second best machine. The costs are determined by assuming that all previously assigned jobs are fixed and all unassigned jobs are irrelevant.

This heuristic still ignores the congestion that results from later assignments, but it does try to identify jobs that have just one attractive machine available. The solution found by this heuristic is termed the *savings-regret* solution.

Once we have a solution, there are a number of ways we might improve on it. For instance, job j might be assigned to machine i but moving it to machine i' decreases the total cost. Such a move is called a 1-exchange. Or it may be that switching j and j' decreases cost. Such a switch is a 2-exchange. We can generalize this to k-exchanges for arbitrary k, but 2-exchanges are sufficient for our purposes. Given a solution, we can do 1- and 2-exchanges until no improvement is possible. This gives a ***locally optimal solution.***

6 Computational Results

In this section we present some computational results which show how well the heuristics and lower bound techniques work. In general, LR-Heur works very well on the problems we generate, but the violated valid inequalities do not affect the lower bound much.

We generate random problems using a generator similar to that used by [15, 16, 10, 4, 8] for the generalized assignment problem (generators A and B). We choose a number of jobs and number of machines. For each

Size (Jobs, Machines)	20, 5			50, 5			50, 10			100, 10		
Range	3	10	20	3	10	20	3	10	20	3	10	20
Greedy	43.7	10.7	11.1	6.6	7.0	3.9	19.9	20.1	12.6	14.9	11.0	6.8
Savings–Regret	29.7	10.4	8.4	7.7	10.1	5.4	21.4	14.9	13.2	12.6	10.6	8.3
LR–Heuristic	5.8	6.3	9.3	4.1	3.9	2.5	13.2	8.5	6.1	5.0	4.1	2.7

Table 1: Percentage of gap remaining.

	20, 5	50, 5	50, 10	100, 10
Greedy	1.5	15.9	14.8	118.9
Savings–Regret	1.5	28.4	25.3	193.4
LR–Heur	3.9	21.8	27.9	104.3

Table 2: Computation time.

Range	3	10	20
Improvement	.5	.3	.8

Table 3: Improvement in lower bound (20 jobs, 5 machines).

assignment, a fixed cost is randomly generated uniformly between 15 and 50. A minimum size is generated uniformly between 5 and 25. We also generate a variable profit between 0 and 5, and a maximum size, generated between 0 and some given range above the minimum size. Finally we have an average target size, and the capacity of each machine is set to the target size times the number of jobs divided by the number of machines. All data is integer.

Our first test compares three heuristics: greedy, savings–regret, and LR–Heur, the heuristic based on the linear relaxation. For this test a target size of 15 (the average minimum size on a machine) is fixed. The range of maximum sizes is either 3 (narrow range), 10 (medium range), or 20 (wide range). We test four sizes of problems: 20 jobs, 5 machines; 50 jobs, 5 machines; 50 jobs, 10 machines; and 100 jobs; 10 machines. Ten problems were run for each upper bound range. The solutions for each heuristic were improved by 1– and 2–exchanges.

Table 1 presents the results. To make the numbers roughly comparable across categories, the results are presented as a percentage of the gap between the lower bound and the greedy solution (without the exchange heuristics). In other words, if the raw greedy value was 50, and the lower bound was 25, a heuristic of value 30 would get value 20% (equals (30-25)/(50-25) * 100). This measure, while slightly arcane, is independent of various data transformations that keep the problem essentially the same. For instance, percentage above lower bound changes if a constant is added to all the costs. The value used does not. Another way to think about this measure is the following: suppose the current method for solving this problem is to use the greedy heuristic without exchanging. This gives a gap above the lower bound. The entry in the table gives the percentage of gap remaining if the raw greedy heuristic is replaced with the alternative heuristics (with exchanging).

Clearly the heuristic based on the linear relaxation does a good job. In only one table entry did the heuristic not find the best solution on average. Furthermore, this average performance is consistent across instances: of the 120 instances represented in the table, LR–Heur had the best solution in 103 cases.

Furthermore, while it is clear that solving generalized networks is a time consuming task, the time required to do the exchanging is considerable. The average times for each problem size (size range had no noticeable effect on computation time) are given in Table 2. Times are in seconds on a Toshiba 5100 (16Mhz 80386 chip, with 80387 math coprocessor, all codes written in Microsoft(c) C version 4.0).

The data structure used to store a solution consisted of a doubly linked list with the jobs sorted in order of their variable cost. Despite this, it takes a tremendous amount of time to do a 2–exchange. This is reflected in the high times for greedy and savings–regret above. LR–Heur finds much better initial solutions so its overall compuatation time is not too large. Note that without exchanging, greedy would have gap 100 (by definition) and savings–regret ended up with gaps ranging from 10 to 80. Exchanging is expensive but it creates much better answers.

The final test determines the effectiveness of the lower bounds generated. Cuts were generated for the smallest problems and added to the problem. The problem was resolved using a linear programming package. In practice, a generalized network with side constraints code ([11]) would be more efficient. Table 3 gives the results. Again, the entries in the table are given in terms of the percentage of the gap between the raw greedy solution and the lower bound (so these values are comparable with those in table 1. It is clear that the lower bound is not increased a lot. Whether this is due to the quality of the linear programming lower bound or the weakness of the constraints cannot be determined from this test.

7 Conclusions

In this paper, we have examined an interesting generalization of standard multiple machine scheduling: variable–speed scheduling. We have shown that the linear relaxation solution contains a lot of information. Most jobs are assigned to only one machine, leaving just a small number to be rescheduled. Furthermore, it is possible to use the linear relaxation to reschedule the rest. Also, it is possible to generate violated valid inequalities from the optimal basis.

Limited computational tests suggest that the resulting heuristic gives good answers consistently, and in a reasonable amount of time. Adding the violated constraints does not increase the lower bound much.

There are a number of other questions to answer for this problem and related problems. First, it is clear that more testing is needed to determine the quality of the heuristic. The smaller problems can probably be solved to optimality, so comparisons with the true value can be made. More sophisticated competitors can be devised.

Second, it is possible to combine the heuristic and the constraint generation in the following iterative algorithm:

(Lagrangian Heuristic)

Step 0) Solve LP2.
Step 1) Find LR–Heuristic solution from current basis.
Step 2) Find violated valid inequality(ies) from current basis and add them to current set of inequalities.
Step 3) Relax current set of inequalities by lagrangian relaxation ([3]), and go to step 1.

The loop can terminate with any standard terminating condition: time limits, iteration limits, convergence, and so on.

Step 1 generates a series of feasible solutions; step 3 generates a series of lower bounds. In general, due to the limited nature of the cuts we generate, we will soon cycle. This is not too critical because solving a lagrangian relaxation each iteration is a very expensive process so we do not wish to do many iterations. Ideally, however, the (non–decreasing) lower bounds and the generation of many feasible solutions may substantially decrease the duality gap.

Third, the violated valid inequalities are generated from one mixed integer constraint. It would be interesting to come up with other valid inequalities that come from more than one constraint. Gottlieb and Rao ([5, 6]) have done this for the generalized assignment problem; the variable–speed scheduling problem is a natural generalization.

Finally, this whole approach suggests that integer generalized networks are an interesting class of mixed integer programs. The network structure leads very naturally to interesting relaxations and heuristics. Furthermore, there is enough structure to make finding violated inequalities easier.

References

[1] G.G. Brown and R. McBride, "Solving generalized networks," *Management Science*, **20:** 1497–1523 (1985).

[2] R.L. Daniels and R.K. Sarin, "Single machine scheduling with controllable processing times and number of jobs tardy," *Operations Research*, **37:** 981–984 (1989).

[3] M.L. Fisher, "An applications oriented guide to lagrangian relaxation," *Interfaces*, 15, 10–21 (1985).

[4] M.L. Fisher, R. Jaikumar, and L. Van Wassenhove, "A multiplier adjustment method for the generalized assignment problem," *Management Science*, 32, 1095–1103 (1986).

[5] E.S. Gottlieb and M.R. Rao, "The generalized assignment problem I: Valid inequalities and facets," Technical report, Graduate School of Business Administration, New York University (1986).

[6] E.S. Gottlieb and M.R. Rao, "The generalized assignment problem II: Three classes of facets," Technical report, Graduate School of Business Administration, New York University (1986).

[7] A.E. Gray, A. Seidmann, and K.E. Stecke, "Tool management in automated manufacturing: operational issues and decision problems," Center for Manufacturing and Operations Management, University of Rochester (1988).

[8] C.D. Jacobs, *The Vehicle Routing Problem with Backhauls*, Ph. D. dissertation, Georgia Institute of Technology (1987).

[9] B. Malakooti and J. Deviprasad, "An interactive multiple criteria approach for parameter selection in metal cutting," *Operations Research*, **37:** 805–818 (1989).

[10] S. Martello and P. Toth, "An algorithm for the generalized assignment problem," *Operational Research '81*, J.P. Brams (editor), North–Holland, New York (1981).

[11] R.D. McBride, "Solving embedded generalized network problems," *European Journal of Operational Research*, 21, 82–92 (1985).

[12] G.L. Nemhauser and L.A. Wolsey, *Combinatorial and Integer Programming*, John Wiley, New York (1988).

[13] W.G. Nulty and M.A. Trick, "GNO/PC generalized network optimization system," *O.R. Letters*, 2, 101–102 (1988).

[14] R.H. Philipson and A. Ravindran, "Application of mathematical programming to metal cutting," *Mathematical Programming*, **23:** 1001–1023 (1979).

[15] G.T. Ross and R.M. Soland, "A branch and bound algorithm for the generalized assignment problem," *Mathematical Programming*, 8, 91–103 (1975).

[16] G.T. Ross and R.M. Soland, "Modelling facility location problems as generalized assignment problems," *Management Science*, 24, 345–357 (1977).

[17] M.A. Trick, "A linear relaxation heuristic for the generalized assignment problem," Carnegie Mellon University (1990).

[18] T.L. Van Roy and L.A. Wolsey, "Valid inequalities for mixed 0–1 programs," *Discrete Applied Mathematics*, **14:** 199–213 (1986).

[19] R.G. Vickson, "Two single–machine sequencing problems involving controllable job processing times," *AIIE Transactions*, **12:** 258–262 (1980).

[20] R.G. Vickson, "Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine," *Operations Research*, **28:** 1155–1167 (1980).

Dual Decomposition of Single Machine Scheduling Problems

S.L. van de Velde
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam
The Netherlands

We propose a framework for Lagrangian duality theory for single machine scheduling with minsum objectives, which induces a dual decomposition of such problems. The dual decomposition method offers promising opportunities for the design of approximative and enumerative algorithms. We develop the method through an analysis of the problem of minimizing the total weighted completion time subject to precedence constraints.

1980 Mathematics Subject Classification (Revision 1985): 90B35.
Keywords & Phrases: single machine scheduling, Lagrangian relaxation, Lagrangian dual, dual decomposition, dynamic programming.

1. Introduction

In the 1980's we have witnessed successes of polyhedral combinatorics for a variety of combinatorial optimization problems. The traveling salesman problem is the subject of several such success stories (see e.g. Lawler et al. 1985). Machine scheduling is a recently discovered challenge for polyhedral techniques (see e.g. Balas 1985, Queyranne and Wang 1988). In view of this, there is renewed interest in the formulation of machine scheduling problems in terms of 0-1 programming. However, the number of variables and constraints needed is a heavy burden for bringing polyhedral techniques into effect. In spite of the initial optimism, polyhedral techniques have not yet been shown to be effective, let alone efficient, beyond the simplest and smallest single machine scheduling problems.

Lagrangian relaxation, on the other hand, is a conventional technique for lower bound computation, that has proved its merits for many types of combinatorial optimization problems. An excellent introduction to Lagrangian relaxation theory and a survey of its applications has been given by Fisher (1981). In the area of machine scheduling, however, Lagrangian relaxation theory is still underdeveloped. The few publications fall into two classes, according to the mathematical formulation applied for the disjunctive constraints that reflect the capacities of the machines.

The first category covers machine scheduling problems that formulate the disjunctive constraints *explicitly*. These applications, however, suffer severely from the number of variables and constraints. Fisher (1976), for instance, introduces a pseudopolynomial number of decision variables x_{jt}, equal to 1 if job J_j completes at time t, and 0 otherwise. Hariri and Potts (1984) and Potts (1985) use decision variables x_{jk} that take the value 1 if job J_j completes before job J_k, of which there are $O(n^2)$ if there are n jobs involved. In this case, however, there

are $O(n^3)$ transitivity constraints needed. On the other hand, the Lagrangian relaxation of each of these formulations is provided with an iterative scheme, tailored after the characteristics of the problem at hand, to find multipliers that solve the Lagrangian dual to optimality or near-optimality.

The second class concerns formulations that present the disjunctive constraints *implicitly*, in which the job completion times are the decision variables. Examples are given by Hariri and Potts (1983) for minimizing total weighted job completion time in the presence of release dates, Potts and Van Wassenhove (1984) for minimizing total weighted completion time with deadlines, and Potts and Van Wassenhove (1985) for minimizing total weighted tardiness. In each of these applications, however, no scheme is proposed to solve the Lagrangian dual. Instead, a specific single pass method is proposed to make an intelligent choice for the multipliers. A notable exception is found in Van de Velde (1988), where the Lagrangian dual is solved for minimizing total completion time in the two-machine flow shop.

This paper is concerned with the second class of formulations for single machine scheduling problems with minsum criteria. For this class, it is possible to develop a duality theory that serves as a framework for the design of both approximative and enumerative methods. In Section 2, we demonstrate this by considering the problem of minimizing total weighted completion time subject to precedence constraints. We consider its Lagrangian relaxation, and propose a method to solve the Lagrangian dual. In addition, we analyze the optimal dual solution and derive some useful properties that we exploit in the design of an approximation algorithm (Section 3) and in an attempt to decompose the primal problem (Section 4). Directions for extensions and future research are given in Section 5.

2. Single machine scheduling

A single machine job shop is described as follows. A set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n independent jobs has to be scheduled on a single machine that can handle only one job at a time. The machine is continuously available from time zero onwards. Each job J_j ($j = 1, \ldots, n$) requires a given positive uninterrupted processing time p_j. In addition, each job J_j has an associated weight w_j, that expresses its urgency relative to other jobs. A *schedule* is a specification of the job completion times, denoted by C_j ($j = 1, \ldots, n$), such that the jobs do not overlap in their execution. The scheduling objective we consider is the minimization of total weighted completion time $\Sigma_{j=1}^{n} w_j C_j$.

This problem, hereafter referred to as problem (P), is formulated as follows. Determine a set of job completion times that minimizes

$$\sum_{j=1}^{n} w_j C_j \tag{P}$$

subject to

$$C_j \geqslant C_k + p_j \quad \text{or} \quad C_j \leqslant C_k - p_k \quad \text{for } j = 1, \ldots, n-1,\ k = j+1, \ldots, n, \tag{1}$$

$$C_j - p_j \geqslant 0 \qquad \text{for } j = 1, \ldots, n. \tag{2}$$

Conditions (1) ensure that the machine processes no more than one job at a time, while conditions (2) reflect that the machine is available from time zero onwards.

PROPOSITION 1. *Problem* (P) *is solved in* $O(n \log n)$ *time by Smith's shortest weighted processing time rule* (Smith, 1956), *which schedules the jobs in order of non-increasing ratios* w_j / p_j.

This rule is easily validated through an interchange argument.

Now, suppose there are precedence constraints between the jobs. Following the notation of Graham et al. (1979), we refer to the problem of minimizing $\Sigma w_j C_j$ subject to precedence constraints on a single machine as $1 \mid prec \mid \Sigma w_j C_j$. The precedence constraints are represented by an acyclic directed graph G with vertex set $\{J_1, \ldots, J_n\}$ and arc set A, which equals its transitive reduction. A path in G from J_j to J_k implies that J_j has to be executed before J_k; J_j is a *predecessor* of J_k, and J_k is a *successor* of J_j. In case there is an arc $(J_j, J_k) \in A$, then J_j is said to be an *immediate predecessor* of J_k; J_k is then an *immediate successor* of J_j. We define $\mathcal{P}_j$ and $\mathcal{S}_j$ as the set of immediate predecessors and immediate successors of J_j, respectively $(j = 1, \ldots, n)$.

The presence of precedence contraints between jobs makes the problem $\mathcal{NP}$-hard (Lawler, 1978, Lenstra and Rinnooy Kan, 1978). This justifies the development of approximative and enumerative algorithms. Morton and Dharan (1978) proposed several heuristics, and Potts (1985) presented a branch-and-bound procedure, based on Lagrangian relaxation of a zero-one programming formulation of the problem. In contrast to Potts, we formulate the precedence constraints in a concise manner. Note that the precedence constraints imply

$$C_k \geqslant C_j + p_k \quad \text{for each } (J_j, J_k) \in A. \tag{3}$$

The $1 \mid prec \mid \Sigma w_j C_j$ problem can be regarded as an easy-to-solve problem complicated by conditions (3). Therefore, we introduce a vector $\lambda \in \mathbb{R}^A$ that contains a Lagrangian multiplier $\lambda_{jk} \geqslant 0$ for each arc $(J_j, J_k) \in A$ and put the constraints (3), each weighted by its multiplier, into the objective function. For given multipliers, the Lagrangian relaxation problem, referred to as problem (L_λ), is to find $L(\lambda)$, which is the minimum of

$$\sum_{j=1}^{n} \Big((w_j + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} - \sum_{J_k \in \mathcal{P}_j} \lambda_{kj}) C_j + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} p_k \Big), \tag{L_λ}$$

subject to conditions (1) and (2).

In analogy to Problem (P), Problem (L_λ) is evidently solved by sequencing the jobs in order of non-increasing values $(w_j + \Sigma_{J_k \in \mathcal{S}_j} \lambda_{jk} - \Sigma_{J_k \in \mathcal{P}_j} \lambda_{kj}) / p_j$. In the remainder, we call these values the *relative weights* of the jobs. From standard Lagrangian theory, we know that $L(\lambda)$ is a lower bound for the $1 \mid prec \mid \Sigma w_j C_j$ problem. In that respect, we are interested in finding values λ_{jk} that maximize $L(\lambda)$. This is the dual problem of $1 \mid prec \mid \Sigma w_j C_j$, referred to as problem (D): maximize

$$L(\lambda) \tag{D}$$

subject to

$$\lambda_{jk} \geqslant 0 \quad \text{for each } (J_j, J_k) \in A.$$

Problem (D) can be transformed into the problem of maximizing a linear function subject to a finite number of linear constraints. There are at most $n!$ feasible sequences involved, and we can represent each of these by a vector of completion times $(C_1^t, C_2^t, \ldots, C_n^t)$, $t = 1, \ldots, T$,

where $T \leqslant n!$; the superscript t refers to the t-th feasible schedule. Problem (D) is then equivalent to the following problem, referred to as problem (D′): maximize

$$v$$

subject to

$$v \leqslant \sum_{j=1}^{n} ((w_j + \textstyle\sum_{J_k \in \mathfrak{S}_j} \lambda_{jk} - \sum_{J_k \in \mathfrak{P}_j} \lambda_{kj}) C_j^t + \sum_{J_k \in \mathfrak{S}_j} \lambda_{jk} p_k) \quad \text{for } t = 1, \ldots, T,$$

$$\lambda_{jk} \geqslant 0 \qquad \text{for each } (J_j, J_k) \in A.$$

PROPOSITION 2. *Problem* (D) *is solvable in time polynomial in n through Khachiyan's ellipsoid method* (Khachiyan, 1979).

PROOF. Let $K = \{(v,\lambda) \in \mathbb{R} \times \mathbb{R}^A \mid v \leqslant \sum_{j=1}^{n} ((w_j + \sum_{J_k \in \mathfrak{S}_j} \lambda_{jk} - \sum_{J_k \in \mathfrak{P}_j} \lambda_{kj}) C_j^t + \sum_{J_k \in \mathfrak{S}_j} \lambda_{jk} p_k)$ for $t = 1, \ldots, T$, $\lambda_{jk} \geqslant 0$, for each $(J_j, J_k) \in A\}$. To prove the proposition, it suffices to show that the following *separation problem for K* is solvable in polynomial time (see Grötschel, Lovász, and Schrijver, 1981, and Padberg and Rao, 1982): given $(\bar{v}, \bar{\lambda}) \in \mathbb{Q} \times \mathbb{Q}^A$, decide whether $(\bar{v}, \bar{\lambda}) \in K$; if not, give a *separating hyperplane*, that is, an inequality $a^T\lambda + \alpha v \leqslant \beta$, such that

$$(v,\lambda) \in K \Rightarrow a^T\lambda + \alpha v \leqslant \beta, \text{ and}$$

$$a^T\bar{\lambda} + \alpha\bar{v} > \beta.$$

Observe now that for given $(\bar{v}, \bar{\lambda})$ we determine the value $L(\bar{\lambda})$ and the corresponding vector $(\bar{C}_1, \bar{C}_2, \ldots, \bar{C}_n)$ of job completion times by solving problem $(L_{\bar{\lambda}})$, which can be done in $O(n \log n)$ time. If $L(\bar{\lambda}) \geqslant \bar{v}$, then $(\bar{v}, \bar{\lambda}) \in K$; if $L(\bar{\lambda}) < \bar{v}$, then $(\bar{v}, \bar{\lambda}) \notin K$, and

$$\bar{v} > \sum_{j=1}^{n} ((w_j + \sum_{J_k \in \mathfrak{S}_j} \bar{\lambda}_{jk} - \sum_{J_k \in \mathfrak{P}_j} \bar{\lambda}_{kj}) \bar{C}_j + \sum_{J_k \in \mathfrak{S}_j} \bar{\lambda}_{jk} p_k),$$

is a separating hyperplane. □

Nonetheless, we propose a different method to solve the dual problem. It follows from the transformation that the function $F: \lambda \to L(\lambda)$ is a concave step-wise linear function in λ. Hence, problem (D) is a convex programming problem that can be solved through an *ascent direction* technique. For given values of the Lagrangian multipliers, we need to determine which component to perturb, i.e. the direction, and by how much, i.e. the step size, in order to increase the value $L(\lambda)$. We have solved the dual problem if no such direction exists. The next algorithm is an ascent direction method that solves problem (D).

ASCENT DIRECTION ALGORITHM

Step 0. Set $\lambda_{jk} = 0$ for each $(J_j,J_k) \in A$, solve (L_λ), and compute the job completion times.

Step 1. For each $(J_j,J_k) \in A$, put $\Delta = 0$ and do the following:

if $C_j > C_k$, though $(J_j,J_k) \in A$, compute Δ such that

$$(w_j + \Delta + \sum_{J_l \in \mathcal{S}_j} \lambda_{jl} - \sum_{J_l \in \mathcal{P}_j} \lambda_{lj})/p_j = (w_k - \Delta + \sum_{J_l \in \mathcal{S}_k} \lambda_{kl} - \sum_{J_l \in \mathcal{P}_k} \lambda_{lk})/p_k;$$

if $C_j < C_k$, $(J_j,J_k) \in A$, and $\lambda_{jk} > 0$, compute the largest value Δ such that

$$\lambda_{jk} - \Delta \geqslant 0, \text{ and}$$

$$(w_j - \Delta + \sum_{J_l \in \mathcal{S}_j} \lambda_{jl} - \sum_{J_l \in \mathcal{P}_j} \lambda_{lj})/p_j \geqslant (w_k + \Delta + \sum_{J_l \in \mathcal{S}_k} \lambda_{kl} - \sum_{J_l \in \mathcal{P}_k} \lambda_{lk})/p_k.$$

In case $\Delta > 0$, adjust λ_{jk} as follows: if $C_j > C_k$, increase λ_{jk} by Δ; if $C_j < C_k$, decrease λ_{jk} by Δ. Accordingly, solve (L_λ) and compute the job completion times.

Step 2. If no multiplier adjustment has taken place, then stop: we have attained the optimal solution. Otherwise, go to Step 1.

PROPOSITION 3. *The procedure described above is an ascent direction method that solves problem* (D) *in a finite number of steps.*

PROOF. First, suppose that $C_j > C_k$ though $(J_j, J_k) \in A$. Let $\Delta > 0$ be the step size that is computed as prescribed in the ascent direction algorithm. In addition, let λ^1 and λ^2 denote the Lagrangian multiplier before and after the adjustment of λ_{jk}, respectively. Hence, the two vectors differ only in one component; for this component we have $\lambda^2_{jk} = \lambda^1_{jk} + \Delta$. This means that the relative weight for J_j is increased, the relative weight for J_k is decreased, and the relative weights for the remaining jobs stay the same with respect to problem (L_{λ^1}). Let the optimal sequence associated with problem (L_{λ^1}) be $(J_1, \ldots, J_{k-1}, J_k, J_{k+1}, \ldots, J_{j-1}, J_j, J_{j+1}, \ldots, J_n)$; we have reindexed the jobs according to increasing completion times. Consequently, the optimal schedule for problem (L_{λ^2}) can be written as $(J_1, \ldots, J_{k-1}, J_{k+1}, \ldots, J_l, J_j, J_k, J_{l+1}, \ldots, J_{j-1}, J_{j+1}, \ldots, J_n)$, for some job J_l with $k+1 \leqslant l \leqslant j-1$. We will now demonstrate that $L(\lambda^2) > L(\lambda^1)$. The vector $(C_1, \ldots, C_n)$ refers to the job completion times in the first schedule; the job completion times for the second schedule can be expressed in terms of this vector and the job processing times. In addition, we let for brevity $\mu_i = w_i + \Sigma_{J_h \in \mathcal{S}_i} \lambda^1_{ih} - \Sigma_{J_h \in \mathcal{P}_i} \lambda^1_{hi}$ for each i, $i = 1, \ldots, n$. Then, we have

$$L(\lambda^2) = \sum_{i=1}^{k-1} \mu_i C_i + \sum_{i=j+1}^{n} \mu_i C_i + \sum_{i=k+1}^{l} \mu_i (C_i - p_k) + \sum_{i=l+1}^{j-1} \mu_i (C_i + p_j) +$$

$$(\mu_k - \Delta)(C_k + p_j + \sum_{i=k+1}^{l} p_i) + (\mu_j + \Delta)(C_j - p_k - \sum_{i=l+1}^{j-1} p_i) + \sum_{i=1}^{n} \sum_{J_h \in \mathcal{S}_i} \lambda^1_{ih} p_h + \Delta p_k$$

$$= L(\lambda^1) + \sum_{i=l+1}^{j-1} (\mu_i p_j - \mu_j p_i) + \sum_{i=k+1}^{l} (\mu_k p_i - \mu_i p_k) + \mu_k p_j - \mu_j p_k +$$

$$\Delta \left[(C_j - p_k - \sum_{i=l+1}^{j-1} p_i) - (C_k + p_j + \sum_{i=k+1}^{l} p_i) \right] + \Delta p_k.$$

Since J_j and J_k are adjacent in the second schedule, we have that

$$(C_j - p_k - \sum_{i=l+1}^{j-1} p_i) - (C_k + p_j + \sum_{i=k+1}^{l} p_i) = -p_k.$$

This implies that

$$L(\lambda^2) = L(\lambda^1) + \sum_{i=l+1}^{j-1} (\mu_i p_j - \mu_j p_i) + \sum_{i=k+1}^{l} (\mu_k p_i - \mu_i p_k) + \mu_k p_j - \mu_j p_k$$

$$= L(\lambda^1) + \sum_{i=k+1}^{l} (\mu_k / p_k - \mu_i / p_i) p_i p_k + \sum_{i=l+1}^{j-1} (\mu_i / p_i - \mu_j / p_j) p_i p_j + (\mu_k / p_k - \mu_j / p_j) p_j p_k.$$

Since $\mu_k / p_k > \mu_j / p_j$, $\mu_i / p_i < \mu_k / p_k$ for each i, $i = k+1, \ldots, l$, and $\mu_i / p_i > \mu_j / p_j$ for each i, $i = l+1, \ldots, j-1$, we have established that $L(\lambda^2) > L(\lambda^1)$. Suppose now that $C_j < C_k$, $(J_j, J_k) \in A$, and $\lambda_{jk} > 0$. If now Δ, computed as described, is greater than 0, then we can perform a similar analysis as above to show that $L(\lambda^2) > L(\lambda^1)$. For that reason, this part of the proof is omitted. Finally, the ascent direction method solves problem (D) in a finite number of steps, since the problem (D) is a linear convex programming problem. □

Notice that

$$C_j \geq C_k \Leftrightarrow (w_j + \sum_{J_l \in \check{S}_j} \lambda_{jl} - \sum_{J_l \in \mathcal{P}_j} \lambda_{lj}) / p_j \leq (w_k + \sum_{J_l \in \check{S}_k} \lambda_{kl} - \sum_{J_l \in \mathcal{P}_k} \lambda_{lk}) / p_k.$$

For each arc $(J_j, J_k) \in A$, we need only constant time to determine the ascent direction, if there exists one. Hence, there is no need to solve (L_λ) and update the completion times in Step 1 of the ascent direction procedure. Since the associated step size is computed in constant time, too, it takes $O(|A|)$ time to verify whether a given set of multipliers solves the dual problem (D). If we have attained the optimal set, then we need $O(n \log n)$ time to compute an optimal dual schedule, the corresponding job completion times, and the optimal dual objective value. Hence, the algorithm runs in $O(I \cdot |A| + n \log n)$ time, where I is the number of iterations. Since I cannot be bounded by a polynomial in n and $|A|$, the ascent direction method is presumably not polynomial. However, from an empirical point of view, it is a fast simplex-like algorithm in the sense that it traverses the edges of the polytope to find the optimal solution.

Consider the 10-job example that is taken from Potts (1985) for which the processing times, weights, and precedence graph are found in Table 1 and Figure 1, respectively.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
p_j	6	9	1	3	9	5	7	7	6	2
w_j	2	5	9	6	5	4	9	3	8	5

TABLE 1

If there were no precedence constraints, which concurs with the problem (L_λ) with λ equal to the null vector, the optimal schedule is $(J_3, J_{10}, J_4, J_9, J_7, J_6, J_2, J_5, J_8, J_1)$, having total cost

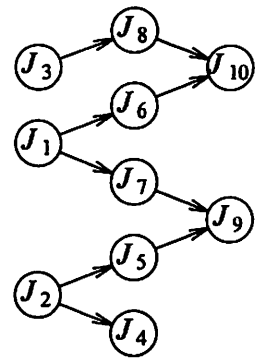


FIGURE 1

1055. This schedule is not feasible for the original problem; for instance, J_{10} is executed before J_6 though $(J_6, J_{10}) \in A$. Accordingly, we compute Δ as prescribed in the ascent direction algorithm; this gives $\Delta = 17/7$, and hence we put $\lambda_{6,10} = 17/7$. This yields $(J_3, J_4, J_9, J_7, J_6, J_{10}, J_2, J_5, J_8, J_1)$ as an optimal schedule for the current Lagrangian problem with $L(\lambda) = 1106$. Proceeding along these lines, we find that the optimal value for problem (D) is 1526.69. In contrast, Potts' procedure, which requires at least $\Omega(n^4)$ time, produces 1519 as a lower bound, while the so-called tree-optimal-heuristic developed by Morton and Dharan generates a schedule with cost 1530, which is the optimal value. Hence, the duality gap is only 3.31.

DEFINITION. The job set $\mathcal{B} \subseteq \mathcal{J}$ is called a *block* if

$$(w_j + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} - \sum_{J_k \in \mathcal{P}_j} \lambda_{kj})/p_j = c \quad \text{for each } J_j \in \mathcal{B},$$

where c is some positive real constant.

Note that the jobs in a block may be interchanged without affecting the dual objective value. For any set of Lagrangian multipliers, the job set $\mathcal{J}$ is decomposed into B blocks $\mathcal{B}_1, \ldots, \mathcal{B}_B$, indexed such that

$$(w_j - \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} + \sum_{J_k \in \mathcal{P}_j} \lambda_{kj})/p_j = c_b, \quad \text{for each } J_j \in \mathcal{B}_b,$$

with $c_1 > \cdots > c_B > 0$. The blocks that are associated with the vector of optimal Lagrangian multipliers (denoted by λ^*) display an agreeable structure.

PROPOSITION 4. *The optimal Lagrangian multipliers decompose the job set $\mathcal{J}$ into B blocks $\mathcal{B}_1, \ldots, \mathcal{B}_B$, such that if $(J_j, J_k) \in A$ and $J_k \in \mathcal{B}_b$, then:*

$$J_j \in \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_b,$$

$$\lambda^*_{jk} = 0 \quad \text{if } J_j \in \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_{b-1},$$

PROOF. If one of these claims were not true, then we could still identify an ascent direction. This contradicts the optimality of the vector λ^*. □

In our example, we obtain three blocks: $\mathcal{B}_1 = \{J_3\}$, $\mathcal{B}_2 = \{J_2, J_4\}$, and $\mathcal{B}_3 = \{J_1, J_5, J_6, J_7, J_8, J_9, J_{10}\}$, with $c_1 = 9$, $c_2 = 11/12$, and $c_3 = 6/7$, respectively.

3. APPROXIMATION

For the optimal Lagrangian multipliers, the decomposition of the job set $\mathcal{J}$ into blocks $\mathcal{B}_1, \ldots, \mathcal{B}_B$ and the properties possessed by these blocks are useful for developing an attractive approximation algorithm. We refer to this decomposition of the job set as the *dual decomposition*. In addition, we speak of *primal decomposition*, if the set of jobs can be decomposed into subsets such that the $1|prec|\Sigma w_j C_j$ problem can be solved to optimality by minimizing total weighted completion time subject to precedence constraints for each subset separately. The issue is how to find such a primal decomposition, since there is no guarantee that the dual decomposition concurs with a primal decomposition. However, the dual decomposition *suggests* a primal decomposition; in this section, we develop an approximation algorithm that is based upon this decomposition. In Section 4, we investigate to which extent the dual decomposition really coincides with a primal decomposition.

Recall that the optimal Lagrangian multipliers decompose the set of jobs into blocks such that for each $(J_j, J_k) \in A$ we have

$$J_k \in \mathcal{B}_b \Rightarrow J_j \in \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_b.$$

If S_b denotes a feasible sequence for the jobs in block $\mathcal{B}_b$, then the schedule S obtained as $S = (S_1, S_2, \ldots, S_B)$ is a feasible solution to the overall problem. Note that we would obtain an approximate solution that is at least as good if for each $\mathcal{B}_b$, S_b would be a sequence for the jobs in $\mathcal{B}_b$ that minimizes the total weighted completion time subject to the precedence constraints that exist between the elements of $\mathcal{B}_b$. Of course, minimizing $\Sigma w_j C_j$ for the block $\mathcal{B}_b$ is as hard as the original problem, but it is of a smaller dimension. For small instances of $1|prec|\Sigma w_j C_j$, a dynamic programming based algorithm with a compact labeling scheme as suggested by Schrage and Baker (1978) and Lawler (1979) is highly efficient. In the approximation algorithm we propose, we minimize $\Sigma w_j C_j$ for a block if less than 15000 labels are needed. Otherwise, we take our resort to the tree-optimal-heuristic developed by Morton and Dharan to generate a feasible sequence for this block. This heuristic has proven to be a very effective, but rather time-consuming approximation algorithm (Morton and Dharan, 1978, Potts, 1985). Our hope is then that the dual decomposition concurs with a primal decomposition, or at least leads up to a promising decomposition, and that the blocks are amenable to dynamic programming.

In the example, the optimal sequences for the first two blocks are trivial: $S_1 = (J_3)$, and $S_2 = (J_2, J_4)$; by dynamic programming (with maximum label 68), we find $S_3 = (J_1, J_7, J_5, J_9, J_6, J_8, J_{10})$, the same sequence as we would have found with the tree-optimal-heuristic. By aggregation of the three sequences, we obtain $S = (J_3, J_2, J_4, J_1, J_7, J_5, J_9, J_6, J_8, J_{10})$ with total cost 1530.

We tested the approximation algorithm on problems with 20, 30, ..., 100 jobs. The processing times were drawn from the uniform distribution [1,100]; the weights were generated from

the uniform distribution [1,10]. The precedence graph was induced by the probability P with which each arc (J_j,J_k) with $j < k$ was included. The graph generated in this way was then subsequently squashed to its transitive reduction. We generated problems for $P = 0.001$, 0.02, 0.04, 0.06, 0.08, 0.10, 0.15, 0.20, 0.30, and 0.50. For each combination of n and P we generated five problems; hence, for each value of P, we generated 45 problems. This procedure parallels Potts' procedure to generate instances.

In Table 2, we present the computational results. As Potts already pointed out, the relative difficulty of an instance is more related to the density of the precedence matrix than to the number of jobs n. Therefore, we have classified the results according to the value P rather than the number of jobs. For each combination of P, we give the proportional deviation between upper bound and lower bound for both the tree-optimal-heuristic and the dual-decomposition approach. Within brackets we indicate for how many problems (out of 45) the upper bound equalled the lower bound, that is, for how many problems we touched upon an optimal solution. As can be seen from this table, the dual-decomposition approach outperforms the tree-optimal-heuristic approach on the average for any problem class. In addition, we found that out of 450 instances, the tree-optimal-heuristic produced only 16 better solutions than the dual-decomposition based algorithm, each of which was only marginally better.

Potts points out that both small and large values for P tend to generate relatively easy problems. For small values there are only few precedence constraints involved, for large values most disjunctive constraints seem to be settled. For small values, this claim is unconditionally supported by our results: the duality gap is very small. As Potts also reports that the tree-optimal-heuristic is a robust procedure for varying values of P, it must be that the duality gap widens if P increases, rather than that the dual decomposition approach subsides.

P	TREE-OPT-HEURISTIC	DUAL DECOMPOSITION
0.001	0.007 (42)	0.007 (42)
0.02	0.074 (15)	0.069 (15)
0.04	0.516 (8)	0.248 (10)
0.06	1.214 (2)	0.675 (2)
0.08	1.446 (1)	1.076 (1)
0.10	2.040 (2)	1.518 (4)
0.15	2.252 (0)	2.024 (0)
0.20	2.551 (0)	2.113 (2)
0.30	4.111 (0)	3.733 (2)
0.50	4.334 (2)	4.116 (3)

TABLE 2: EXPERIMENTAL RESULTS

The tree-optimal-heuristic requires $O(n^3)$ time, and is therefore sensitive to instances with a large number of jobs. The running time of the dual decompositon approximation algorithm mainly depends on the number of calls on the dynamic programming algorithm and the maximum label number. We have coded both algorithms in the computer language C; all

experiments were conducted on a Compaq 386/20 Personal Computer. For $n \leqslant 40$, the tree-optimal-heuristic generally needed a few seconds at the most. Though for these values for n our approximation required only slightly more computation time on the average, there were occasional peaks as a result of a relatively high labels in the dynamic programming subroutine. For $n \geqslant 60$, we found out that the tree-optimal-heuristic needed about twice or thrice as much computation time as the dual-decomposition algorithm; even the peaks for the latter remained below the average of the former.

4. Primal Decomposition

The dual decomposition only suggests a primal decomposition; however, there are instances that demonstrate that the dual decomposition may eliminate the optimal solution. In this section, we derive sufficient conditions to establish to which extent the dual composition coincides with a primal decomposition. If the dual decomposition excludes any optimal solution, then there must be at least two jobs belonging to different blocks for which the processing order should be reversed. Consider the jobs $J_j \in \mathfrak{B}_b$ and $J_k \in \mathfrak{B}_{b+m}$ $(m > 0)$ for which there is no path in A from J_j to J_k. Suppose that J_k precedes J_j in any optimal schedule. This would imply that the arc (J_k, J_j) could be included in the arc set A without impunity. Let now $\lambda^*(k,j)$ denote the optimal Lagrangian multiplier for problem (D_{kj}), which is the dual problem with respect to the set A augmented with the arc (J_k,J_j). Accordingly, $L(\lambda^*(k,j))$ is the optimal objective value for the problem (D_{kj}). Since J_j and J_k belong to different blocks, we must have that $L(\lambda^*(k,j)) > L(\lambda^*)$. If $L(\lambda^*(k,j)) > UB - 1$, where UB is the currently best solution to the $1\,|prec|\,\Sigma w_j C_j$ problem in hand, then we know that in any solution with cost less than UB, if such a solution exists, J_j must be executed before J_k. Based upon this observation, we have the following result.

Proposition 5. *If for each pair of jobs $J_j \in \mathfrak{B}_b$ and $J_k \in \mathfrak{B}_{b+m}$, $b = 1, \ldots, B-1$, $m = 1, \ldots, B-b$, such that*

(i) there is no path in A from J_j to J_k,

(ii) J_j has no successors scheduled in $\mathfrak{B}_b \cup \cdots \cup \mathfrak{B}_{b+m-1}$, and

(iii) J_k has no predecessors scheduled in $\mathfrak{B}_{b+1} \cup \cdots \cup \mathfrak{B}_{b+m}$,

we have that $L(\lambda^*(k,j)) > UB - 1$, then the dual decompositon is a primal decomposition.

Accordingly, if the dual decomposition induces a primal decomposition in the sense of Proposition 5 and if UB is associated with the schedule that is composed of the optimal schedules for each block separately, then UB is the optimal solution value to the $1\,|prec\,|\,\Sigma w_j C_j$ problem.

Corollary 1. *If for some block $\mathfrak{B}_b$ each pair of jobs $J_j \in \mathfrak{B}_b$ and $J_k \in \mathfrak{B}_{b+m}$, $m = 1, \ldots, B-b$, such that*

(i) there is no path in A from J_j to J_k,

(ii) J_j has no successors scheduled in $\mathfrak{B}_b \cup \cdots \cup \mathfrak{B}_{b+m-1}$, and

(iii) J_k *has no predecessors scheduled in* $\mathfrak{B}_{b+1} \cup \cdots \cup \mathfrak{B}_{b+m}$,

satisfies $L(\lambda^*(k,j)) > UB - 1$, *then the set* $\mathfrak{J}$ *can be primally decomposed into the subsets* $\mathfrak{B}_1 \cup \cdots \cup \mathfrak{B}_b$ *and* $\mathfrak{B}_{b+1} \cup \cdots \cup \mathfrak{B}_B$.

In this case, we say that the dual decomposition concurs *partly* with a primal decomposition. From the next proposition, it follows that $\lambda^*(k,j)$ is conveniently computed from λ^*.

PROPOSITION 6. *We have that*

$$\lambda^*(k,j)_{ih} = \lambda^*_{ih} \text{ if either } J_i \text{ or } J_h \notin \mathfrak{B}_b \cup \cdots \cup \mathfrak{B}_{b+m}.$$

PROOF. The introduction of the arc (J_k,J_j) only affects the jobs in the blocks $\mathfrak{B}_b, \ldots, \mathfrak{B}_{b+m}$. Hence, we can only have that $\lambda^*(k,j)_{ih} \neq \lambda^*_{ih}$ if both J_i and $J_h \in \mathfrak{B}_b \cup \cdots \cup \mathfrak{B}_{b+m}$. □

Nonetheless, the verification of Proposition 5 demands the solution of $O(n^2)$ reoptimization problems. It is conceivable that if J_j and J_k belong to blocks that lie far apart from each other, there is no need to go through the entire reoptimization procedure. It may suffice to perform only the first step in the ascent direction procedure. This very first step can be conveniently computed; this is stipulated in the next proposition, where P_b is defined as $P_b = \Sigma_{J_i \in \mathfrak{B}_b} p_i$.

PROPOSITION 7. *If there are two jobs* $J_j \in \mathfrak{B}_b$ *and* $J_k \in \mathfrak{B}_{b+m}$, $b = 1, \ldots, B-1$, $m = 1, \ldots, B-b$ *such that there is no path in* A *from* J_j *to* J_k *for which*

$$L(\lambda^*) + (c_b - c_{b+m})p_j p_k + \sum_{i=b+1}^{l} (c_b - c_i)P_i p_j + \sum_{i=l+1}^{b+m-1} (c_i - c_{b+m})P_i p_k > UB - 1,$$

where l *is the largest index with* $c_l \geqslant (p_j c_b + p_k c_{b+m})/(p_k + p_j)$, *then* J_j *precedes* J_k *in any optimal solution to the* $1 \mid prec \mid \Sigma w_j C_j$ *problem.*

PROOF. The validation of this proposition, which requires the same logic as applied in the proof of Proposition 3, is left to the reader. □

We now work out the effects of these propositions on our example. To decompose the jobs into $\mathfrak{B}_1$ on one side and $\mathfrak{B}_2 \cup \mathfrak{B}_3$ on the other, we need consider only the pairs (J_3,J_2) and (J_3,J_1) according to Corollary 1. If we include (J_2,J_3) in A, then it suffices to execute only one step in the reoptimization procedure. The application of Proposition 7 yields that $L(\lambda^*) + (c_1 - c_2)p_2 p_3 = 1526.69 + (9 - 11/12)\cdot 1\cdot 9 > 1529$, as a result of which we conclude that J_3 precedes J_2 in any schedule with cost less than 1530. Similarly, if we include (J_1,J_3) in A, then, according to Proposition 7 (where we have $l = 1$), we must check if

$$L(\lambda^*) + (c_1 - c_3)p_1 p_3 + (c_2 - c_3)P_2 p_1 > UB - 1.$$

It is easy verifiable that this is true; hence, J_3 must precede J_1, which implies that we may decompose the job set into subsets $\mathfrak{B}_1$ and $\mathfrak{B}_2 \cup \mathfrak{B}_3$. We must consider the pairs (J_4,J_1), (J_4,J_5), and (J_4,J_8) to separate the blocks $\mathfrak{B}_2$ and $\mathfrak{B}_3$. For the associated reoptimization

problems more than one step is required. Since $L(\lambda^*(1,4))$, $L(\lambda^*(5,4))$, and $L(\lambda^*(8,4))$ are greater than 1529, we conclude that the dual decomposition induces a complete primal decomposition. Furthermore, as the schedule with value 1530 was obtained from the optimal sequences for the individual blocks, it must be an optimal schedule.

The propositions that are presented in this section are applicable in a preprocessing phase in conjunction with any existing branch-and-bound algorithm. Their main purpose is then to derive additional precedence constraints and to primally decompose the problem in order to reduce the size of the branch-and-bound tree.

5. Conclusions

Dual theories similar to the one discussed here for the $1|prec|\Sigma w_j C_j$ problem can be developed for other single machine scheduling problems with minsum criteria functions provided that the Lagrangian problem reduces to problem (P). However, the effectiveness of the dual decomposition approach depends on the nature of the relaxed constraints. In this respect, the two most eye-catching and likely applications are the problems of minimizing the total weighted completion time subject to deadlines and minimizing the total weighted tardiness. For the latter problem, for instance, it would be interesting to see to which extent the multiplier adjustment method presented by Potts and Van Wassenhove (1985) could be integrated with or replaced by our approach. In addition, the various approximation algorithms for this problem evaluated by Potts and Van Wassenhove (1988) ask for a comparison with the dual decomposition method.

Acknowledgment

The author likes to thank Bert Gerards and Jan Karel Lenstra for their helpful suggestions.

References

E. Balas (1985). On the facial structure of scheduling polyhedra. *Mathematical Programming Study 24*, 179-218.

M.L. Fisher (1976). A dual algorithm for the one-machine scheduling problem. *Mathematical Programming 11*, 229-251.

M.L. Fisher (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science 27*, 1-18.

R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics 5*, 287-326.

M. Grötschel, L. Lovász, and A. Schrijver (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica 1*, 169-197. [Corrigendum (1984): *Combinatorica 4*, 291-295.]

A.M.A. Hariri and C.N. Potts (1983). An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics 5*, 99-109.

A.M.A. Hariri and C.N. Potts (1984). Algorithms for two-machine flow-shop sequencing with precedence constraints. *European Journal of Operational Research 17*, 238-248.

L.G. KHACHIYAN (1979). A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR 244*, 1093-1096 (English translation: *Soviet Mathematics Doklady 20*, 191-194).

E.L. LAWLER (1978). Sequencing jobs to minimize total weighted subject to precedence constraints. *Annals of Discrete Mathematics 2*, 75-90.

E.L. LAWLER (1979). *Efficient Implementation of Dynamic Programming Algorithms for Sequencing Problems*, Report BW 106, Centre for Mathematics and Computer Science, Amsterdam.

E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN AND D.B. SHMOYS (eds.) (1985). *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

J.K. LENSTRA AND A.H.G. RINNOOY KAN (1978). Complexity of scheduling under precedence constraints. *Operations Research 26*, 22-35.

T.E. MORTON AND B.G. DHARAN (1978). Algoristics for single-machine sequencing with precedence contraints. *Management Science 24*, 1011-1020.

M.W. PADBERG AND M.R. RAO (1982). Odd minimum cut-sets and *b*-matchings. *Mathematics and Operations Research 7*, 67-80.

C.N. POTTS (1985). A Lagrangean based branch-and-bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Science 31*, 1300-1311.

C.N. POTTS AND L.N. VAN WASSENHOVE (1983). An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *European Journal of Operational Research 33*, 363-377.

C.N. POTTS AND L.N. VAN WASSENHOVE (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research 33*, 363-377.

C.N. POTTS AND L.N. VAN WASSENHOVE (1988). *Single machine tardiness sequencing heuristics*, Report 8906/A, Erasmus University, Rotterdam.

M. QUEYRANNE AND Y. WANG (1988). *Single machine scheduling polyhedra with precedence constraints*, Working Paper No. 88-MSC-017, University of British Columbia, Vancouver.

L. SCHRAGE AND K.R. BAKER (1978). Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research 26*, 444-449.

W.E. SMITH (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly 3*, 59-66.

S.L. VAN DE VELDE (1990). Minimizing the sum of the job completion times in the two-machine flow shop by Lagrangian relaxation. To appear in *Annals of Operations Research.*

A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{VE})$ General Graph Matching Algorithm

Vijay V. Vazirani

Computer Science Department
Cornell University

1. Introduction

Finding a maximum matching in a graph is a classical problem in the study of algorithms. In this paper we present new algorithmically relevant combinatorial structure of matchings. This structure yields the first proof of correctness of the general graph matching algorithm of Micali and Vazirani [MV]; this is currently the most efficient known matching algorithm.

Berge's theorem [Be], which says that matching M in graph G is maximum iff there are no augmenting paths w.r.t to it, gives an iterative schema for finding a maximum matching in G, i.e. successively find augmenting paths. Finding augmenting paths is fairly easy in bipartite graphs; however, not so in general graphs (see [LP] for a detailed history of the problem). The first polynomial time algorithm ($O(|V|^4)$) for general graph matching was given by Edmonds [Ed]. In this paper, Edmonds introduced the notion of a *blossom* (an odd length alternating cycle), and showed that by 'shrinking' blossoms, one can find an augmenting path efficiently. In this seminal paper, Edmonds also introduced the notion of a polynomial time algorithm.

Over the years, several authors have given faster implementations of Edmonds' algorithm (Gabow [Ga], Kameda and Munro [KM], Lawler [La]). In 1972, Hopcroft and Karp [HK] proposed finding augmenting paths in *phases*; in each phase a maximal set of disjoint minimum length augmenting paths is found. They showed that only $O(\sqrt{|V|})$ phases are needed, as opposed to $O(|V|)$ iterations in the previously-mentioned schema. They also presented an $O(|E|)$ implementation of a phase in bipartite graphs, thereby giving an $O(\sqrt{|V|}\,|E|)$ matching algorithm for such graphs, and left the open problem of obtaining an algorithm having the same efficiency for general graphs. Using the idea of phases, an $O(|V|^{2.5})$ algorithm for general graphs was given by Even and Kariv [EK]. The algorithm of [MV] achieves the above-stated $O(\sqrt{|V|}\,|E|)$ running time on the RAM model; it uses the incremental-tree set union algorithm of Gabow and Tarjan [GT1] which runs in linear time on a RAM.

In giving the proof of correctness, our emphasis is on formally laying out algorithmically relevant structure of matchings. Once this is done (in Theorems 1 to 7), the rest of the proof follows quite easily. The terminology established in stating the structural results also leads to a simple description of the algorithm. Central to the structure is a definition of blossoms from the perspective of minimum length alternating paths, and a study of properties of these paths w.r.t. the blossoms. However, the structure is very rich, to the extent that considerable preparation is needed before blossoms can even be defined. For this reason, we will give an overview of the structure in Section 2 after introducing some definitions.

Three important notions underlie this theory: tenacity, base, and blossom. The following remarks are worth mentioning: We have been unable to find a simple, intuitive definition of tenacity. Yet, from it emerges, in an intuitively clear manner, the notion of base of a vertex. This notion, together with tenacity, leads to the definition of blossoms and a proof of correctness of the algorithm. However, the algorithm does not explicitly find blossoms or the base of a vertex, instead it finds petals and buds (a blossom is the union of

its petals). On the other hand, base and blossom appear to be essential for stating the combinatorial structure on which the algorithm is required to operate, and for giving its proof of correctness.

The algorithm contains two main ideas: the precise manner in which the various events are synchronized, and the graph searching procedure of double depth first search (DDFS). The correctness of DDFS and the synchronization are also established (in Theorems 8 and 9 respectively).

In [GT2] Gabow and Tarjan give an efficient scaling algorithm for weighted matching; its cardinality version is related to the algorithm in [MV] and achieves the same time bound. It is interesting to observe that besides cardinality matching, for several other matching problems, such as weighted matching, finding a maximum matching in parallel, and approximately computing the number of perfect matchings in dense graphs, the known algorithms for general graphs require additional ideas but achieve the same efficiency as for bipartite graphs. Is this just a coincidence, or is there an underlying reason for this?

2. Overview of the Structural Results and the Algorithm

The following definitions are standard. Let $G(V,E)$ be a graph. A set $M \subseteq E$ is said to be a *matching* if every vertex of G has at most one edge of M incident at it. M is a *maximum matching* if it is a matching of largest possible cardinality in G. The following terms are defined w.r.t. a matching M in G: edges in M are said to be *matched*, and those in $E-M$ are *unmatched*. A vertex is said to be *matched* if it has a matched edge incident on it, and *unmatched* otherwise; sometimes an unmatched vertex is also referred as a *free* vertex. If (u,v) is a matched edge, then we say that *u is the matched neighbour of v*. A simple path is said to be an *alternating path* if it consists alternately of matched and unmatched edges. An *augmenting path* is an alternating path that starts and ends at (distinct) unmatched vertices.

Let us first consider the bipartite case. Given a bipartite graph $G(U, V, E)$ with matching M, define the *level* of vertex $x \in U \cup V$ to be the length of a shortest alternating path from an unmatched vertex in U to x. Notice that vertices in U have even levels and those in V have odd levels, and among the unmatched vertices in V, the one having smallest level gives the length of a minimum length augmenting path w.r.t. M. The natural schema for finding the levels of vertices is an alternating breadth first search (BFS) (i.e. seek unmatched and matched edges at alternating levels), starting at the unmatched vertices in U. This schema works, leading to a simple $O(|E|)$ implementation for a phase.

The above-stated schema fails in general graphs. We need the following definitions from [MV] to illustrate this:

Definition: W.r.t. matching M in graph $G(V,E)$ define:
evenlevel (v): Length of the shortest even length alternating path from an unmatched vertex to v; ∞ if no such path exists.
oddlevel (v): Length of the shortest odd length alternating path from an unmatched vertex to v; ∞ if no such path exists.
maxlevel (v): The larger of *evenlevel*(v) and *oddlevel*(v).
minlevel (v): The smaller of *evenlevel*(v) and *oddlevel*(v).

The levels of vertices are marked in figure 1. If there is a minimum length augmenting path between unmatched vertices f_1 and f_2 then f_1 and f_2 have the smallest oddlevel among all unmatched vertices in G. So let us first consider the problem of finding the evenlevel and oddlevel of all vertices. The alternating BFS fails in figure 1: if it is not allowed to go around edge (u,v) then w does not get its maxlevel, and if it is, then b gets an incorrect maxlevel. The algorithm of [MV] gets around this by 'searching down' from u and v to find b (this is done by DDFS). In order to state it precisely, we will need the following definitions.

Definition: W.r.t. matching M in graph $G(V,E)$ define (the first two definitions appear in [MV]):
tenacity (v) = *evenlevel*(v) + *oddlevel*(v)
For edge (u,v),
tenacity (u, v) = *evenlevel*(u) + *evenlevel*(v) +1 if (u,v) is unmatched

$oddlevel(u)+oddlevel(v)+1$ if (u,v) is matched.

A path that gives v its evenlevel will be called an *evenlevel*(v) *path*. The terms *oddlevel*(v) *path*, *maxlevel*(v) *path* and *minlevel*(v) *path* are similarly defined.

predecessor: if (u,v) is the last edge on a *minlevel*(v) path then u is *predecessor* of v.

prop: edge (u,v) s.t. either u is a predecessor of v or v is a predecessor of u.

bridge: An edge which is not a prop.

For a bridge (u,v),

$$support(u,v) = \{w \mid tenacity(w) = tenacity(u,v), \text{ and there is a } maxlevel(w) \text{ path containing } (u,v)\}$$

Remark: It is tempting to define *tenacity*(v) to be the length of a shortest alternating walk between two (not necessarily distinct) unmatched vertices which contains v. However, this is not true. For example, vertices v and b in figure 1 will have the same tenacity under this definition.

These notions are illustrated in figure 2. The tenacities of vertices are marked. The graph has three bridges (j,k), (l,m) and (n,o); the remaining edges are props. These bridges have tenacities 11, 13 and 15 respectively, and their supports are $\{j,k\}$, $\{h,i,l,m\}$ and $\{c,d,e,g,n,o\}$ respectively.

Here is an overview of the algorithm. The algorithm is iterative; at each search level (starting with search level 0), it executes one step of BFS (procedure MIN), followed by DDFS (procedure MAX). MIN finds minlevels and MAX finds maxlevels of vertices in the following manner:

At search level i:

MIN: Finds the minlevel of the following set of vertices $\{v \in V \mid minlevel(v)=i+1\}$.

The set of bridges, $B_{2i+1}=\{(u,v)\in E \mid (u,v)$ is a bridge and $tenacity(u,v)=2i+1\}$ is also found.

MAX: Finds the maxlevel of the following set of vertices $\{v \in V \mid tenacity(v)=2i+1\}$
These vertices are found as follows: for each bridge (u,v) in B_{2i+1}, call DDFS with (u,v) to find *support*(u,v).

Procedure MIN is straightforward: if i is even (odd), it simply searches along all unmatched (matched) edges incident at vertices u such that $evenlevel(u) = i$ ($oddlevel(u) = i$). Notice that the evenlevel (oddlevel) of u will be known at this stage: if this is the minlevel of u, then MIN would have found it during search level $i-1$; otherwise $tenacity(u) \leq 2i-1$, and MAX would have found it during search level $(tenacity(u)-1)/2$. On the other hand, MAX is not so straightforward; in order to prove its correctness, we need to show:

1). All bridges of tenacity $2i+1$ are found at the end of the execution of MIN during search level i.

2). When called with bridge (u,v), DDFS finds *support*(u,v).

3). For every vertex v, every *maxlevel*(v) path contains a bridge having tenacity = *tenacity*(v).

Remark: A vertex may be in the support of more than one bridge. However, this fact is not important at present; it will be handled in Section 11.

We will consider fact (3) first; the structural results presented in Sections 3 to 8 will eventually help prove this fact. Below we present an intuitive description of this structure.

If v has finite tenacity then *base*(v) is that unique vertex which is the highest vertex of *tenacity* > *tenacity*(v) on any *evenlevel*(v) or *oddlevel*(v) path (the uniqueness is proved in Theorem 3). The bases of various vertices in figure 2 are:

j,k: g
h,i,l,m: e
c,d,e,g,n,o: b

The blossom having *base* b and *tenacity* t, $B_{b,t}$, is intuitively the set of vertices of *tenacity* $\leq t$ that 'sit on' b. Thus,

$B_{g,11} = \{j,k\}$
$B_{e,13} = \{h,i,l,m\}$
$B_{b,15} = \{c,d,e,g,h,i,j,k,l,m,n,o\}$

In figure 2, vertices are placed according to their minlevels; this reveals the blossom structure clearly. Notice that $B_{g,11} \subseteq B_{b,15}$ and $B_{e,13} \subseteq B_{b,15}$; this is the nesting structure of blossoms. Our central structural result is that every *evenlevel*(v) (*oddlevel*(v)) path consists of an *evenlevel*(*base*(v)) path concatenated with a minimum even (odd) length alternating path from *base*(v) to v through the blossom $B_{base(v),tenacity(v)}$. The *minlevel*(v) path through the blossom goes 'directly' from *base*(v) to v, and the *maxlevel*(v) path goes 'around the blossom', i.e. uses the bridge; this is proven in Theorem 7, and leads to fact (3). For example, the two paths from e to h are: $e\ h$ and $e\ i\ m\ l\ h$, and those from b to e are : $b\ c\ e$ and $b\ d\ g\ j\ k\ o\ n\ l\ h\ e$.

The fact that the *maxlevel*(v) path contains the bridge is of algorithmic significance: it shows how v can get its maxlevel by 'searching down' from the two endpoints of its bridge. DDFS is a natural search schema to do this efficiently; it grows, in a coordinated manner, two DFS trees rooted at the two endpoints of the bridge, and finds the support of the bridge. Fact (2) follows from Theorem 7 and the correctness of DDFS, and is established in Proposition 1.

Notice how the various events are synchronized in the algorithm: At search level i, the BFS step finds the minlevels of all vertices having minlevel $i+1$, and DDFS finds the maxlevels of all vertices having *tenacity* $2i+1$ (rather than maxlevel $i+1$!). Fact (1) is essential for synchronization and is proven in Theorem 9. In turn, for ensuring fact (1), special kinds of bridges, called anamolies (defined in Section 9), need to be handled appropriately in the algorithm. In Section 12 we present examples to show why things go wrong if the events are not synchronized in the above-stated manner.

While presenting the structural theorems we will assume that G has only one unmatched vertex f; this simplifies some of the arguments. Using the following simple transformation, the results carry over to arbitrary graphs (this is shown at the end of Section 8): given a graph, add a new matched edge at each unmatched vertex, and connect the other endpoints of these edges via unmatched edges to a single unmatched vertex f.

3. Tenacity and Breadth-First-Search Honesty

In bipartite graphs, minimum length alternating paths are *breadth–first–search–honest* in the following sense: $G(U,V,E)$ is a bipartite graph with matching M, $x \in U \cup V$, p is a *level*(x) path starting at unmatched vertex $f \in U$, and y is a vertex on p. Then the part of p from f to y is a *level*(y) path.

On the other hand, minimum length alternating paths are not BFS honest in general graphs. For example in figure 2, *evenlevel*(c) = 12, and vertex l occurs at a distance of 9 on the *evenlevel*(c) path. However, *oddlevel*(l) = 7.

In Theorems 1 and 2 we will use the notion of tenacity to prove that even in non-bipartite graphs, minimum length alternating path are BFS honest to some extent, namely, higher tenacity vertices on the path occur at the 'correct' distance. Theorem 1 deals with a *minlevel*(v) path and Theorem 2 with a *maxlevel*(v) path.

Lemma 1: Let (u,v) be a matched edge. Then, *evenlevel*(v) = *oddlevel*(u) +1 and *evenlevel*(u) *oddlevel*(v) +1. Also, *tenacity*(u) = *tenacity*(v) = *tenacity*(u,v).

Proof: Any alternating path containing both u and v must contain the matched edge (u,v). The first equality follows by observing that any *oddlevel*(u) path cannot contain v, and therefore concatenating (u,v) to it yields an *evenlevel*(v) path. Adding *oddlevel*(v) to both sides of this equality we get *tenacity*(v) = *tenacity*(u,v). The remaining equalities follow in a similar manner. □

Notation: If p and q are two paths, $p \bar{} q$ denotes their concatenation, and $|p|$ denotes the length of path p.

Definition: Let p be alternating path starting at an unmatched vertex f, and let u and v be two vertices occurring on p (in either order, u before v or v before u). Then $p[u \text{ to } v]$ denotes the part of p from u to v (both

inclusive). Similarly $p[u \text{ to } v)$, $p(u \text{ to } v]$ and $p(u \text{ to } v)$ denote the part of p from u to v, including u only, including v only, and excluding both u and v, respectively. We will say that *u occurs at the correct distance on p* if:
(*i*). if $|p[f \text{ to } u]|$ is even, then $|p[f \text{ to } u]| = evenlevel(u)$
(*ii*). if $|p[f \text{ to } u]|$ is odd, then $|p[f \text{ to } u]| = oddlevel(u)$.
Similarly, *u occurs at $minlevel(u)$ distance on p* if $|p[f \text{ to } u]| = minlevel(u)$, and *$u$ occurs at $maxlevel(u)$ distance on p* if $|p[f \text{ to } u]| = maxlevel(u)$.
Vertex u is an *even (odd) vertex w.r.t. p* if $|p[f \text{ to } u]|$ is even (odd), and *v is higher than u on p* if $|p[f \text{ to } v]| > |p[f \text{ to } u]|$.

Theorem 1: Let p be a $minlevel(v)$ path and let u be a vertex on p such that $tenacity(u) \geq tenacity(v)$. Then u occurs at $minlevel(u)$ distance on p.

Proof: It is sufficient to prove the theorem for $|p|$ odd, because if $|p|$ is even, consider $p[f \text{ to } v)$ which is a minlevel path to the matched neighbour of v. We will prove that if u does not occur at the correct distance on p then $tenacity(u) < tenacity(v)$. By Lemma 1, w.l.o.g. we may assume that u is even w.r.t. p. Let q be an $evenlevel(u)$ path. We will use q to show that $oddlevel(u) < minlevel(v)$. Since $evenlevel(v) < minlevel(v)$ also, this will show that $tenacity(u) < tenacity(v)$. The situation is illustrated in figure 3.

Vertex q must intersect $p(u \text{ to } v]$, because otherwise $q \bar{} p[u \text{ to } v]$ is a shorter odd path to v. Let v' be the matched neighbour of v, and let $p' = p \bar{} (v,v')$. Let w be the first vertex of q on $p'(u \text{ to } v']$. If w is odd w.r.t. p', then once again, by splicing parts of q and p, we can get a shorter odd path to v. Therefore, w is even w.r.t. p'. Now, $q[f \text{ to } w] \bar{} p'[w \text{ to } u]$ is an odd length alternating path from f to u, giving the desired inequality $oddlevel(u) < |q| + |p'[u \text{ to } w]| < |p|$.

Hence, if $tenacity(u) \geq tenacity(v)$, then u must occur at the correct distance, in fact $minlevel(u)$ distance on p. □

Theorem 2: Let p be a $maxlevel(v)$ path, and u be a vertex on p such that $tenacity(u) \geq tenacity(v)$. Then,
(i) if $tenacity(u) = tenacity(v)$, u occurs at the correct distance on p
(ii) if $tenacity(u) > tenacity(v)$, u occurs at $minlevel(u)$ distance on p.

Proof: By Lemma 1, w.l.o.g. we may assume that $|p|$ is even and that u is even w.r.t. p. (Notice that unlike in Theorem 1, if u occurs at the correct distance on p, it does not follow that u occurs at $minlevel(u)$ distance. For this reason, we first consider the $minlevel(v)$ path in order to establish a relationship between $minlevel(u)$ and $minlevel(v)$.)

Let q be a $minlevel(v)$ path. If q does not intersect $p[u \text{ to } v)$, then $oddlevel(u) \leq |q| + |p[v \text{ to } u]|$. Clearly, $evenlevel(u) \leq |p[f \text{ to } u]|$. Therefore $tenacity(u) \leq tenacity(v)$. Since we have assumed that $tenacity(u) \geq tenacity(v)$, it follows that $tenacity(u) = tenacity(v)$ and u occurs at the correct distance on p.

Next suppose that q intersects $p[u \text{ to } v)$. Let u' be the matched neighbour of u, and let w be the first vertex of q on $p[u' \text{ to } v)$. Vertex w must be odd w.r.t. p, because otherwise there is a short odd length alternating path to u, showing $tenacity(u) < tenacity(v)$. Now, $|q[f \text{ to } w]| \geq |p[f \text{ to } w]|$, because otherwise by splicing q and p we can get a shorter even path to v. Therefore $|q| = minlevel(v) > evenlevel(u)$. Since $tenacity(u) \geq tenacity(v)$, the minlevel of u must be its evenlevel. It remains to show that u must occur at the correct distance on p. The argument is the same as in Theorem 1: if not, then the $evenlevel(u)$ path enables us to prove that $tenacity(u) < tenacity(v)$. □

The next lemma follows from the proof of Theorem 2.

Lemma 2: Let p be a $maxlevel(v)$ path and u be a vertex on p such that $tenacity(u) > tenacity(v)$. Then $minlevel(v) > minlevel(u)$.

4. The Base of a Vertex

In this section we will use the notion of tenacity to define the base of a vertex; this will eventually be the base of the blossom in which the vertex lies.

Definition: Let v be a vertex of finite tenacity, and let p be an *evenlevel*(v) or an *oddlevel*(v) path. The *base of* v *w.r.t.* p, denoted by *base*(v,p), is the highest vertex on p having tenacity greater than that of v (there must be such a vertex since *tenacity*(f) = ∞).

The main result of this section is to show that the base of v is unique, i.e. independent of p: Towards this end we will first show that if p and q are *evenlevel*(v) and *oddlevel*(v) paths respectively, then *base*(v,p) = *base*(v,q). Let b be the highest vertex of *tenacity* > *tenacity*(v) occurring on both p and q. The proof involves showing that all vertices on $p(b$ to $v]$ are of *tenacity* ≤ *tenacity*(v). This is done by studying the intersections of p and q; for this we will define flowers, and show that the intersections of p and q form flowers.

Definition: Let p be an alternating path starting at f. An odd length alternating path that meets p only at its endpoints, and starts and ends with unmatched edges is called a *segment w.r.t.* p. A set of segments w.r.t. p satisfying certain conditions form a *flower w.r.t.* p, F. The base (tip) of F is the lowest (highest) vertex of p which is on one of these segments. The flower F will be the union of these segments together with p[*base*(F) to *tip*(F)]. The vertices on this part of p are said to be *covered* by F. Following is a recursive definition of the conditions which the set of segments should satisfy:

(i). the set consists of a single segment that starts and ends at even vertices w.r.t. p.

(ii). let F' be a flower and q be a segment one of whose endpoints is covered by F', and the other endpoints is even w.r.t. p. Then the set of segments of F' together with q form a flower.

(iii). let F' and F'' be flowers, and q be a segment whose two endpoints are covered by F' and F'' respectively. Then the sets of segments of F' and F'' together with q form a flower.

The *length of flower* F, denoted by $|F|$, is defined to be the sum of the lengths of the segments of F and $|p[base(F) \text{ to } tip(F)]|$. Figure 4 shows a flower formed by segments q_1, q_2, q_3 and q_4.

The above-stated recursive definition of flower yields the following lemma by a straightforward induction.

Lemma 3: Let F be a flower w.r.t. path p, and v be even (odd) w.r.t. p and covered by F. If $v \neq base(F)$, then there is an odd (even) length alternating path in F from *base*(F) to v of length ≤ $|F|$.

Definition: Let p be an alternating path starting at f, and let q be any other alternating path. Then, a part of q that starts and ends with unmatched edges and meets p only at its endpoints is called a *segment of* q *w.r.t.* p. A flower w.r.t. p formed by segments of q is said to be a *flower of* q. An alternating path whose first and last edges are matched edges on p (the rest of the path may also intersect p), is called a *section w.r.t.* p.

Lemma 4: Let p be an alternating path starting at f and q be a section w.r.t. p which starts and ends at vertices s and s' on p. Then at least one of the following must hold:

(i). s is even w.r.t. p, or s is covered by a flower of q.

(ii). s' is even w.r.t. p, or s' is covered by a flower of q.

Proof: The proof is by an induction on the number of segments in q. The assertion is obvious in case q consists of no segments. We prove the induction step below.

Suppose s and s' are both odd w.r.t. p. There are two cases. First, suppose there is a segment of q that starts at a vertex above s and ends at a vertex below s. Let q' be the first such segment, and let y and y' be its starting and ending vertices. Since s is not covered by a flower of $q[s \text{ to } y]$, by the induction hypothesis, y is either even w.r.t. p or covered by a flower of $q[s \text{ to } y]$. Now, if y' is even w.r.t. p, s is covered by a flower of $q[s \text{ to } y']$. So, assume that y' is odd w.r.t. p. If s' is covered by a flower of $q[y' \text{ to } s']$, we are done. Otherwise, by the induction hypothesis, y' is covered by a flower of $q[y' \text{ to } s']$. Now using the segment q', s is covered by a flower of q. The last case is illustrated in figure 5.

Next, suppose there is no such segment q'. Let z be the highest vertex of p on q. Since z is even w.r.t. p, $z \neq s'$. Now, $q[s' \text{ to } s]$ satisfies the first case, and by the proof given above, s' is covered by a flower of q (since clearly s is not). □

For the next two lemmas, let v be a vertex of finite tenacity, and let p and q be *evenlevel*(v) and *oddlevel*(v) paths respectively. Consider vertices of *tenacity* > *tenacity*(v) which occur on both p and q (f is such a vertex), and let b be the highest such vertex. (Recall that such vertices occur at their minlevel distance on both p and q.) We will first prove that *base*$(v,p) =$ *base*$(v,q) = b$ in a simple setting: when there are no separators. Edges (s,s') in figure 10 (a) and (b) are separators.

Definition: We will say that matched edge (w,w') is a *separator w.r.t. p* and q if it occurs on both p and q, and is a cut edge for the subgraph formed by $p \cup q$. For example, in figure 9(a), (s,s') is a separator w.r.t. *evenlevel*(v) and *oddlevel*(v) paths.

Remark: The matched edge incident at b is a separator w.r.t. p and q. This fact follows from Lemma 7; however, we do not need it for proving Theorem 3.

Lemma 5: If there are no separators w.r.t. p and q on $p(b \text{ to } v]$ (and therefore also on $q(b \text{ to } v]$) then *base*$(v,p) =$ *base*$(v,q) = b$.

Proof: We will prove that every vertex on $p(b \text{ to } v]$ has *tenacity* $\leq$ *tenacity*(v); the proof for vertices on $q(b \text{ to } v]$ is similar. For this, it is sufficient to show that for vertex u on $p(b \text{ to } v]$ which is even w.r.t. p, there is an odd length alternating path of length $\leq$ *tenacity*$(v) - |p[f \text{ to } u]| = |q| + |p[u \text{ to } v]|$. We will first need the following definitions: let (w,w') be a matched edge occurring on both $p(b \text{ to } v]$ and $q(b \text{ to } v]$, with w' even w.r.t. p. If before traversing (w,w'), q does not meet any vertex of p higher than w', then (w,w') is called a *frontier*. If w' is odd w.r.t. q, (w,w') is called a *backward frontier*. If (w,w') is not a separator w.r.t. p and q, and w' is even w.r.t. q then (w,w') is called a *forward frontier*. Backward and forward frontiers are illustrated in figures 6 and 7 respectively.

Let matched edge (w,w') on p be the closet frontier to u such that $|p[f \text{ to } w']| \geq |p[f \text{ to } u]|$, where w' is even w.r.t. p. If (w,w') is a backward frontier, then $|p[f \text{ to } w']|$ $\bar{}p[w' \text{ to } u]$ is the required path (see figure 6). Next consider the case that (w,w') is a forward frontier. Clearly $|q[f \text{ to } w']| \geq |p[f \text{ to } w']|$, because otherwise by splicing p and q we can get a shorter even path to v.

We will first prove that w is covered by a flower of $q[w \text{ to } v]$. Consider the last segment of $q[w \text{ to } v]$ which starts below w (say at z) and ends above w. Now, z couldn't be even w.r.t. p because otherwise $p[f \text{ to } z]\bar{}q[z \text{ to } v]$ is a shorter odd path to v. If z is covered by a flower of $q[w \text{ to } z]$, then this flower must cover w also because otherwise we can again get a shorter odd path to v using Lemma 2. In the last case, by Lemma 3, w must be covered by a flower of $q[w \text{ to } z]$. Let b' be the base of this flower, and let r be an even length alternating path from b' to w in this flower.

Now, if u is above b', then $p[f \text{ to } b']$ concatenated with the odd length alternating path from b' to u through the flower gives the odd path to u. Otherwise, $q[f \text{ to } w]\bar{}r[w \text{ to } b']\bar{}p[b' \text{ to } u]$ is the required path. The first case is illustrated in figure 7 and the second in figure 8. □

Lemma 6: *base*$(v,p) =$ *base*$(v,q) = b$.

Proof: The no separators case is proved in Lemma 5. Let (s,s') be the lowest separator on $p(b \text{ to } v]$ and $q(b \text{ to } v]$, with s' even w.r.t. p and q. Clearly, $|p[f \text{ to } s]| = |q[f \text{ to } s]|$, and by the choice of b, *tenacity*$(s) \leq$ *tenacity*(v). Let (w,w') be the highest frontier on $p(b \text{ to } s)$, with w' even w.r.t. p. By the

proof of Lemma 5, all vertices on $p(b \text{ to } w']$ have *tenacity* $\leq$ *tenacity*(v).

For dealing with vertices on $p(w' \text{ to } s)$, first consider the case that s occurs at the correct distance on p (e.g. see figure 9(a)). Let r be an *evenlevel*(s) path which shares the most number of vertices with q. If r has no separators on $p(w' \text{ to } s)$, then by the proof of Lemma 5, all vertices on this part of p have *tenacity* $\leq$ *tenacity*(v). Otherwise, let (x,x') be the highest separator, with x' even w.r.t. p. Once again, the proof of Lemma 5 takes care of vertices on $p(x' \text{ to } s)$. For the remaining vertices, there are some cases to be considered.

The canonical case is when $r[x' \text{ to } s]$ does not intersect $q[f \text{ to } s)$. Let u be an even vertex on $p(w' \text{ to } x']$. We will show that the odd path $p' = q[f \text{ to } s]^{\frown} r[s \text{ to } x']^{\frown} p[x' \text{ to } u]$ has length $\leq$ *tenacity*$(v) - |p[f \text{ to } u]|$, thereby bounding *tenacity*(u).

Since s occurs at the correct distance on p, $|r[f \text{ to } s]| \leq$ *tenacity*$(v) - |p[f \text{ to } s]|$. Also, $|r[f \text{ to } x']| \geq |p[f \text{ to } x']|$, because otherwise we can splice p and r to get a shorter even path to v. Therefore, $|r[x' \text{ to } s]| \leq$ *tenacity*$(v) - |p[f \text{ to } s]| - |p[f \text{ to } x']|$. Substituting this in $|p'|$ and using $|p[f \text{ to } s]| = |q[f \text{ to } s]|$ gives the bound.

Next suppose $r[x' \text{ to } s]$ intersects $q[f \text{ to } s]$ in vertex y first. If y is even w.r.t. q then by the same argument as above, $q[f \text{ to } y]^{\frown} r[y \text{ to } x']^{\frown} p[x' \text{ to } u]$ is the required odd path to u. Finally, suppose y is odd w.r.t. q. Then, $|r[f \text{ to } y]| \geq |q[f \text{ to } y]|$. Now, $r[y \text{ to } s]$ must intersect $q[f \text{ to } y)$, because otherwise r violates the condition that it shares the most number of vertices with q. Let (z,z') be the lowest matched edge of $q[f \text{ to } y)$ traversed by $r[y \text{ to } s]$, with z' even w.r.t. q. If z' is even w.r.t. r, then we can get a shorter even path to s by splicing r and q. Otherwise, $q[f \text{ to } z']^{\frown} r[z' \text{ to } x']^{\frown} p[x' \text{ to } u]$ is the required odd path to u.

In case s does not occur at the correct distance on p (e.g. see figure 9(b)), let r be an *oddlevel*(s) path. Now, r must intersect $p(s \text{ to } v]$ in an even vertex first. Let this vertex be h, and as before, let (x,x') be the highest separator of r on $p(w' \text{ to } s)$. Let $r' = r[x' \text{ to } h]^{\frown} p[h \text{ to } s]$. Using r' in place of $r[x' \text{ to } s]$ in the above-stated cases yields the required odd path to u.

Finally, we remark that these arguments apply to vertices between any two consecutive separators on p; the vertices between the highest separator on p and v are dealt with using the proof of Lemma 5. □

Theorem 3: Let v be a vertex of finite tenacity. Then its base is unique, i.e. the set $\{b \mid b = base(v,p)$ for some *evenlevel*(v) or *oddlevel*(v) path $p\}$ is a singleton.

Proof: Let p and q be any *evenlevel*(v) and *oddlevel*(v) paths. By lemma 5, $base(v,p) = base(v,q) = b$ (say). Now, by fixing p and varying q over all *oddlevel*(v) paths and then fixing q and varying p over all *evenlevel*(v) paths we get required result. □

Definition: For a vertex v of finite tenacity define $base(v)$ to be its unique base. Say that a vertex v is *outer* if *evenlevel*(v)<*oddlevel*(v), and *inner* if *oddlevel*(v)< *evenlevel*(v).

Remark: 1). For a matched edge (u,v), $base(u) = base(v)$, by Lemma 1.
2). For a vertex v of finite tenacity, $base(v)$ is an outer vertex.

Definition: Let v be a vertex of finite tenacity. Define $base^1(v) = base(v)$. Furthermore, for $k \in Z^+$, if $base^k(v)$ is of finite tenacity, then define $base^{k+1}(v) = base\ (base^k(v))$.

Remark: Notice that *tenacity* $(base^{k+1}(v)) >$ *tenacity* $(base^k(v))$, and *evenlevel* $(base^{k+1}(v)) <$ *evenlevel* $(base^k(v))$.

Corollary 1: Let v be a vertex such that $base^{k+1}(v)$ exists, for $k \in Z^+$, and let p be any *evenlevel*(v) or *oddlevel*(v) path. Then every vertex on $p(base^{k+1}(v) \text{ to } v]$ has *tenacity* $\leq$ *tenacity* $(base^k(v))$.

5. The Significance of Base

We will use the notion of base to define blossoms in the next section. The other significance of base is that every *evenlevel*(v) path consists of an *evenlevel* (*base*(v)) path concatenated with a minimum even-alternating path from *base*(v) to v. Thus the two paths can be found *independently*. A similar statement holds for *oddlevel*(v) paths.

Definition: An *even–alternating path* (*odd–alternating path*) from u to v, is an alternating path of even (odd) length, starting with an unmatched edge.

Lemma 7: Let u be a vertex occurring on an *evenlevel*(v) (*oddlevel*(v)) path p. Suppose u is even w.r.t. p and *tenacity*(u) > *tenacity*(v). Let q be an *evenlevel*(u) path and r be a minimum length even-alternating (odd-alternating) path from u to v. Then q and r meet only at u.

Proof: Suppose not, and let (w,w') be the lowest matched edge of q traversed by r, with w' even w.r.t. q. If w' is even w.r.t. r, than by splicing parts of q and r, we can get an even (odd) path to v which is shorter than $|q| + |r|$. However, by Theorems 1 and 2, $|p| \geq |q| + |r|$, leading to a contradiction. Suppose w' is odd w.r.t. r. Then $q[f \text{ to } w'] \bar{} r[w' \text{ to } b]$ is an odd path to b of length $< |p|$. We now get that *tenacity*(u) < *tenacity*(v) (in case p is a *minlevel*(v) path this is obvious, otherwise use Lemma 2). The contradiction proves the lemma. □

Remark: We have considered only the case that u is even w.r.t. p. This is so because of the manner in which we defined even-alternating and odd-alternating paths: they always start with an unmatched edge. The reason for this choice will become clear in Theorem 4.

Notice that in general u may not occur on every *evenlevel*(v) path, e.g. see figure 10.

Theorem 4: Let v be a vertex of finite tenacity, and let $b = base(v)$. Then, every *evenlevel*(v) (*oddlevel*(v)) path consists of an *evenlevel*(b) path concatenated with a minimum length even-alternating (odd-alternating) path from b to v.

Proof: Follows from Lemma 7 and Theorem 3.

Definition: Let $b = base^k(v)$, for $k \in Z^+$. Define *evenlevel*(v,b) (*oddlevel*(v,b)) to be the length of a shortest even-alternating (odd-alternating) path from b to v. The smaller of these two is called *minlevel*(v,b), and the larger is called *maxlevel*(v,b).

Corollaries 2 and 3 follow from Theorems 1 to 4.

Corollary 2: Let $b = base(v)$, and let p be an *evenlevel*(v,b) or an *oddlevel*(v,b) path. Let u be even w.r.t. p with *tenacity*(u) = *tenacity*(v). Then, $p[b \text{ to } u]$ is an *evenlevel*(u,b) path. Moreover, if p is a *minlevel*(v,b) path then $p[b \text{ to } u]$ is a *minlevel*(u,b) path.

Corollary 3: Let v be a vertex such that $base^1(v)$, $base^2(v) \cdots base^k(v)$ exist. Let p_k be an *evenlevel*($base^k(v)$) path, and p_l be an *evenlevel*($base^l(v)$, to $base^{l+1}(v)$) path, for $1 \leq l \leq k-1$. Finally, let p_0 be an *evenlevel*($v,base^1(v)$) (*oddlevel*($v,base^1(v)$)) path. Then $p_k \bar{} p_{k-1} \cdots \bar{} p_1 \bar{} p_0$ is an *evenlevel*(v) (*oddlevel*(v)) path, and $p_{k-1} \bar{} \cdots \bar{} p_1 \bar{} p_0$ is an *evenlevel*($v,base^k(v)$) (*oddlevel*($v,base^k(v)$)) path.

6. Blossoms and their Significance

In this section we will define blossoms from the perspective of minimum length alternating paths. Theorem 5 gives the central result that all shortest alternating paths from *base*(v) to v lie in a blossom.

Definition: Let v be a vertex of finite tenacity, and let t be an odd positive integer such that $t \geq tenacity(v)$. Let $k = \min\{j \in Z^+ \mid tenacity((base^j(v)) \geq t\}$, and $l = \min\{j \in Z^+ \mid tenacity(base^j(v)) > t\}$. Define $base_{\geq t}(v) = base^k(v)$, and $base_{>t}(v) = base^l(v)$.

Remark: Let p be an *evenlevel*(v) or *oddlevel*(v) path. Then by Corollary 1, every vertex on $p(base_{\geq t}(v)$ to $v]$ is of *tenacity* $< t$, and every vertex on $p(base_{>t}(v)$ to $v]$ is of *tenacity* $\leq t$.

Definition: Let b be an outer vertex, and t be an odd positive integer such that $t < tenacity(b)$. The *blossom of tenacity t having base b*;

$$B_{b,t} = \{v \in V \mid tenacity(v) \leq t \text{ and } base_{>t}(v) = b\}.$$

In general the vertices of a blossom may not even be connected. For example, in figure 10, $B_{b,13} = \{a,c,d,e,v,g\}$.

Remark: 1). If Edmonds' algorithm is modified to 'shrink' sets of vertices in stages: at stage i, shrink all vertices of tenacity $2i+1$, then 'macronodes' obtained at the end of each stage correspond exactly to the blossoms defined above.

2). For matched edge (u,v), u and v belong to the same blossoms.

We will need the following properties of blossoms to prove Theorem 5.

Lemma 8: Let B_1 and B_2 be two blossoms. Then either they are disjoint or one is contained in the other.

Proof: Let B_1 be a blossom of tenacity t_1 and base b_1, and B_2 be a blossom of tenacity t_2 and base b_2. Let $t_1 \leq t_2$. Suppose $v \in B_1 \cap B_2$. Then $base_{>t_1}(v_1) = b_1$ and $base_{>t_2}(v_2) = b_2$. If $b_1 = b_2$, then clearly $B_1 \subseteq B_2$

Consider the case $b_1 \neq b_2$. Then, $base_{>t_2}(b_1) = b_2$. Let u be any vertex in B_1. Then $tenacity(u) \leq t_1$ and $base_{t_1}(u) = b_1$. Therefore, $base_{>t_2}(u) = b_2$. Therefore, $u \in B_2$. Hence $B_1 \subseteq B_2$. □

The proof of the following lemma is straightforward.

Lemma 9: Let v be a vertex such that $base^1(v) = b_1$, $base^2(v) = b_2 \cdots base^k(v) = b_k$. Let $tenacity(v) = t_0$, and $tenacity(b_i) = t_i$, for $1 \leq i \leq k$. Let B_i be the blossom of tenacity t_{i-1} having base b_i, for $1 \leq i \leq k$. Then, $B_1 \subseteq B_2 \subseteq \cdots \subseteq B_k$.

Lemma 10: Let $b = base^k(v)$, for $k \in Z^+$. Let p be an *evenlevel*(b) path, and let $u \neq b$ be even w.r.t. p. Then (u,v) is not an edge in the graph.

Proof: If (u,v) is an edge, there is an odd path to v of length less then *evenlevel*(b), giving a contradiction. □

Theorem 5: Let v be a vertex of finite tenacity. Let $b = base(v)$, $t = tenacity(v)$, and $B_{b,t}$ be the blossom having base b and tenacity t. Let p be an *evenlevel*(v,b) or an *oddlevel*(v,b) path. Then any vertex on $p(b$ to $v]$ is in $B_{b,t}$.

Proof: By Lemma 1, it is sufficient to prove the theorem for $|p|$ even. The proof is by induction on *evenlevel*(v). For the base case, let v be a vertex of finite tenacity having the smallest evenlevel. Then clearly $|p| = 2$, and since the matched neighbour of v is in $B_{b,t}$, the assertion holds. We prove the induction step below.

Suppose $p(b$ to $v]$ contains a vertex not in $B_{b,t}$. Let u be the highest such vertex on p; clearly u is even w.r.t. p. First consider the case $|p[u$ to $v]| > 2$. Let v' be the matched neighbour of v, and let w be the highest vertex on $p(u$ to $v')$. Now, w couldn't be of tenacity t because otherwise by Corollary 2, $p[b$ to $w]$ is an *evenlevel*(w,b) path, which contradicts the induction hypothesis. Therefore, *tenacity*$(w) < t$. Let $b' = base_{\geq t}(w)$, and let q be an *evenlevel*(w,b') path. If $b' = b$, $q^{-}(w,v')^{-}(v',v)$ is a shorter even-alternating path from b to v (using Corollary 3, Lemma 9 and the induction hypothesis). Otherwise,

tenacity$(b')=t$ and $b' \in B_{b,t}$. Let r be an *evenlevel*(b',b) path. Vertex v cannot be on r (if it is even w.r.t. r, we get a shorter path from b to v, otherwise this contradicts Lemma 10) or on $q(b'$ to $w]$ (because these vertices are of tenacity $<t$). Now, by the induction hypothesis, $r \,^{-}q\,^{-}(w,v')\,^{-}(v',v)$ is a shorter path from b to v.

Finally, consider the case $|p[u \text{ to } v]|=2$. Let $b'=base_{>t}(u)$, and let q be an *evenlevel*(u,b') path and r be an *evenlevel*(b') path. By the induction hypothesis, $q(b'$ to $u]$ is in the blossom having base b' and tenacity t, and therefore $v \notin q$. As before, v cannot be on r. Since b is not $base^k(v)$ for $k \in Z^+$, *evenlevel*$(b) + |p[b \text{ to } u]| >$ *evenlevel*$(u) = |r| + |q|$. Therefore, $r \,^{-}q\,^{-}(u,v')\,^{-}(v',v)$ is a shorter even path to v than that obtained by concatenating an *evenlevel*(b) path with p. The contradiction proves the induction step. □

The following feature of blossoms is being used in the above-stated inductive proof: that p is contained in a blossom, and therefore has a simple interface to the rest of the graph, through the base of the blossom.

7. The Nesting of Blossoms

The nesting of blossoms is described in Lemma 11. Notice that in figure 2, the *evenlevel*(c,b) path either enters or exists from blossoms nested in $B_{b,15}$ at their base (i.e. vertices g and e), and each blossom is used at most once. This is established in Theorem 6. As a consequence, the part of this path in the nested blossom is a minimum length alternating path; this is shown in Corollary 4. These properties of paths w.r.t. the nested blossom structure reveal the reason for BFS dishonesty.

Lemma 11: Let $B_{b,t}$ be a blossom of tenacity t having base b. Let $C=\{v \mid tenacity(v)=t$ and $base_{>t}(v)=b\}$. For $u \in C \cup \{b\}$, let $B_u=\{v \mid tenacity(v)<t$ and $base_{\geq t}(v)=u\}$. Then $B_{b,t}=C \cup (\bigcup_{u \in C \cup \{b\}} B_u)$.

Proof: Let $v \in B_{b,t}$. If $tenacity(v)=t$, then $base(v)=b$ and $v \in C$. Suppose $tenacity(v)<t$. Then $base^k(v)=b$, for some positive integer k. If $k=1$, $v \in D_b$. Otherwise, let $u=base^{k-1}(v)$. Clearly, $tenacity(u) \leq t$. If $tenacity(u)=t$, then $u \in C$, and $v \in D_u$. If $tenacity(u)<t$, then $v \in D_b$. Containment in the other direction is obvious.□

Definition: Let B_1 and B_2 be two blossoms. If B_1 is a proper subset of B_2 then we will say that B_1 is *nested* in B_2. If furthermore there is no blossom B_3 such that $B_1 \subseteq B_3 \subseteq B_2$ then B_1 is *properly nested* in B_2. The *nesting depth* of blossom B is defined recursively as follows: if B has no properly nested blossoms then its nesting depth is 0. Otherwise, among the blossoms properly nested in B, let B' have the largest nesting depth, and let k be the nesting depth of B'. Then the nesting depth of B is $k+1$.

Lemma 12: Let $u \in C$ as defined in Lemma 11, and $v \in B_u$. Then,
minlevel$(v,b) >$ *evenlevel*(u,b), and
maxlevel$(v,b) <$ *oddlevel*(u,b).

Proof: The proof follows from Corollary 2, and the fact that *tenacity*$(v) <$ *tenacity*(u). □

Theorem 6: Let v be a vertex of tenacity t in blossom $B_{b,t}$, and p be an *evenlevel*(v,b) or an *oddlevel*(v,b) path. Then p enters and exits from any blossom properly nested in $B_{b,t}$, say B_u, at most once. If so, p must either enter or exit from the set $B_u \cup \{u\}$ at vertex u.

Proof: It is sufficient to prove the theorem for $|p|$ even. The proof is by induction on *evenlevel*(v,b) for vertices of tenacity t in $B_{b,t}$. For the base case, let v be such a vertex having smallest *evenlevel*(v,b). Clearly, any vertex of tenacity $<t$ on p must be in the nested blossom B_b. Let w be the highest such vertex on p. Then by Theorem 5, $p(b$ to $w]$ is in B_b, proving the assertion. We prove the induction step below.

Suppose p does not satisfy the condition, and let B_u be the last blossom which p enters and 'uses improperly'. The proof of the base case shows that $u \neq b$. Let w be the first vertex of p in $B_u \cup \{u\}$, and w' be the last. If either $w=u$ or $w'=u$, then by Theorem 5 we can get a shorter even-alternating path from b to v which uses B_u 'properly'. Otherwise, let q be an *evenlevel*(u,b) path; by Lemma 12, $|q| < |p|$, and therefore by the induction hypothesis q satisfies the condition. By Corollary 2 any vertex of tenacity t must be at its correct distance on q; moreover since u is outer, this must be its minlevel distance. Therefore, q must enter a set $B_{u'} \cup \{u'\}$ at u'. Assume that $p(b \text{ to } w]$ and $p[w' \text{ to } v]$ both intersect q; the remaining cases are simpler. A vertex of tenacity t on $q \cap p[w' \text{ to } v]$ must occur at its maxlevel distance on p. Now, if $B_{u'}$ is a blossom such that u' occurs on q then $p[w' \text{ to } v]$ must enter $B_{u'}$ at most once (since B_u is the last 'misused' blossom on p), and must exit the set $B_{u'} \cup \{u'\}$ at u'. Let x be the first vertex of $p[w' \text{ to } v]$ which is on q and has tenacity t. By Theorem 5, $q[x \text{ to } u]$ concatenated with an *evenlevel*(w,u) path is shorter than $p[x \text{ to } w]$. Therefore, if $p(b \text{ to } w]$ does not intersect $q[x \text{ to } u]$, we can get a shorter even path from b to v. Otherwise, let y be the first vertex of $p(b \text{ to } w]$ on $q[x \text{ to } u]$. First assume *tenacity*$(y) = t$. If y is odd w.r.t. q, then since $q[y \text{ to } u]$ concatenated with an *evenlevel*(w',u) path is shorter than $p[y \text{ to } w']$, we can get a shorter path to v. Otherwise we can use $q[y \text{ to } x]$ instead of $p[y \text{ to } x]$. Next suppose *tenacity*$(y) < t$, and $y \in B_{u'}$. Then, $p[x \text{ to } y]$ is an even length alternating path from x to y. But the shortest such path is obtained by concatenating $q[x \text{ to } u']$ with an *evenlevel*(y,u') path (this path does not use B_u). Again, this gives a shorter path to v. The contradiction proves the induction hypothesis. □

Remark: If p is a *minlevel*(v,b) path then all properly nested blossom used by p are entered through the base. On the other hand suppose p is *maxlevel*(v,b) path and uses nested blossoms $B_1 \cdots B_k$ in this order. Then, either all of these blossoms are entered through the base or all are exited through the base, or $\exists i$, $1 \leq i < k$ such that $B_1 \cdots B_i$ are entered through the base and $B_{i+1} \cdots B_k$ are exited through the base.

Corollary 4: Suppose p enters (exits) the set $B_u \cup \{u\}$ at u and exits (enters) at w. Then $p[u \text{ to } w]$ is an *evenlevel*(w,u) path and therefore *evenlevel*$(w) =$ *evenlevel*$(u) + |p[u \text{ to } w]|$.

Proof: Follows from Theorems 5 and 6, and the minimality of p.

Remark: Theorem 6 and Corollary 4 are also true for any blossom nested (not necessarily properly) in $B_{b,t}$. This can be proven by an easy induction on the nesting depth of $B_{v,t}$.

We can now explain why minimum length alternating paths are BFS dishonest in general graphs. Let p be a *maxlevel*(v,b) path that enters $B_u \cup \{u\}$ at vertex $w \neq u$. By Theorem 4 and 5, every *oddlevel*(w,b) path consists of an *evenlevel*(u,b) path concatenated with an *oddlevel*(w,u) path; the latter path is contained in B_u. Therefore $|p[b \text{ to } w]| >$ *oddlevel*(w,b). However, this is consistent with Theorems 1 and 2 since *tenacity*$(w) <$ *tenacity*(v). The following corollary complements Theorems 1 and 2 for the case *tenacity*$(w) <$ *tenacity*(v), and gives additional constraints that minimum length alternating paths must satisfy despite their BFS dishonesty. It can be proven by an easy induction on $|p|$.

Corollary 5: Let p be an *evenlevel*(v) or an *oddlevel*(v) path, and w be a vertex of finite tenacity on p. Then *base*(w) is also on p; moreover, $p[$*base*$(w) \text{ to } w]$ is an *evenlevel*$(w,$*base*$(w))$ or an *oddlevel*$(w,$*base*$(w))$ path, depending on the parity of $|p[$*base*$(w) \text{ to } w]|$.

8. Every Maxlevel Path Contains a Bridge

We will first prove that every edge on the path from *base*(v) to v has *tenacity* $\leq$ *tenacity*(v). This is followed by the last structural theorem: that a *maxlevel*(v) path contains a unique bridge of tenacity *tenacity*(v). Theorem 7 (b) shows how v can be obtained by searching down from the bridge; this fact will eventually be used in DDFS. Theorem 7(c) gives a relationship between the even and oddlevels of the endpoints of the bridge and *tenacity*(v). This is used in Lemma 15 to prove that bridges are found 'well in time' to make the synchronization work.

Lemma 13: All edges on p have tenacity $\leq t$.

Proof: By induction on the nesting depth of $B_{b,t}$, for the base case, suppose $B_{b,t}$ has nesting depth 0. Then every vertex on $p(b \text{ to } v]$ has tenacity t. So by Lemma 1, every matched edge on p has tenacity t. Let (u,u') be an unmatched edge on p, with u even w.r.t. p. Since u and u' occur at the correct distance on p, and $tenacity(u') = t$, it follows that $evenlevel(w') = t - evenlevel(u) - 1$. This gives $tenacity(u,u') = t$.

We now prove the induction step. Suppose p enters (exits) the set $B_u \cup \{u\}$ in u and exits (enters) in w. Then, by Corollary 4, and the induction hypothesis, all edges on $p[u \text{ to } w]$ have tenacity $<t$. Of the remaining edges on p, the tenacity of matched edges is t since they are incident on vertices of tenacity t. Finally consider a remaining unmatched edge (u,u'). If both endpoints have tenacity t (or more, since b may be an endpoint), then u and u' occur at the correct distance on p, and the argument of the base case applies. Otherwise suppose $tenacity(u) = t$ and $tenacity(u') < t$. Let $b' = base_{\geq t}(u')$; by Theorem 6, b' must be on p. Now, using Corollary 4 and the fact that b' must be at the correct distance on p, it is easy to see that $tenacity(u,u') = t$. This proves the induction step. □

Theorem 7: Let v be a vertex of finite tenacity, say t, and let p be a $maxlevel(v)$ *path*. Then,

(a): There is a unique bridge of tenacity t *on* p.

Proof: We will first show the existence of such a bridge and then its uniqueness. Let $b = base(v)$, and let set S consist of all vertices on $p[b \text{ to } v]$ which have tenacity $\geq t$. By Theorem 2, these vertices occur at the correct distance on p. Partition S into two sets: $S_{\min}$ ($S_{\max}$) consists of vertices of S which occur at their minlevel (maxlevel) distance on p. Notice that $b \in S_{\min}$ and $v \in S_{\max}$. Let w be the highest vertex of p in $S_{\min}$, and let w' be the lowest vertex of p in $S_{\max}$. Notice that all vertices of $S_{\min}$ lie on $p[b \text{ to } w]$ and those of $S_{\max}$ on $p[w' \text{ to } v]$.

First suppose (w,w') is a matched edge. Clearly, $oddlevel(w) = |p[f \text{ to } w]|$ and $oddlevel(w') = t - |q[f \text{ to } w']|$, giving $tenacity(w,w') = t$. Moreover, since the minlevel of both w and w' is odd, (w,w') is not a prop, and is therefore a bridge.

In the remaining case, w and w' are both outer vertices, and all vertices on $p(w \text{ to } w')$, if any, are of tenacity $<t$. By Theorem 6, these vertices must be either in B_w or $B_{w'}$, i.e. blossoms of tenacity $t-2$ having base w and w' respectively. Let x be the highest vertex of $p(w \text{ to } w')$ in B_w; if there is no such vertex, let $x = w$. Let x' be the first vertex on $p(x \text{ to } w']$. clearly (x,x') is unmatched; we will show that (x,x') is a bridge of tenacity t.

By Corollary 4, $evenlevel(x) = |p[f \text{ to } x]|$, and $evenlevel(x') = evenlevel(w') + |p[w' \text{ to } x']|$. Also, $evenlevel(w') = t - |p[f \text{ to } w']|$. This gives $tenacity(w,w') = t$. Now, x' is not a predecessor of x; if $x=w$, the predecessor of x is its matched neighbour, and otherwise the predecessor of x is in B_w. Similarly x is not a predecessor of x', and so (x,x') is a bridge.

We finally prove uniqueness. Let (u,u') be an edge on $p[b \text{ to } w]$ with u' higher than u. If $tenacity(u') = t$, u is a predecessor of u', and so (u,u') is a prop. Otherwise by Lemma 13, (u,u') is of tenacity $<t$. The same applies for an edge (u,u') on $p[w' \text{ to } v]$ with u higher than u'. Also, by Lemma 13, edges on $p[w \text{ to } w']$ other than (x,x'), if any, are of tenacity $<t$. Finally consider edge (u,u') on $p[f \text{ to } b]$, with u' higher then u. If $tenacity(u') \geq t$, u is a predecessor of u'. Otherwise by Corollary 5, $base_{\geq t}(u')$ must be on $p[f \text{ to } u')$, and so by Lemma 13, $tenacity(u,u') < t$. □

The following definition and extensions of Theorem 7(a) give algorithmically useful facts; their proof follows from the proof of Theorem 7(a).

Definition: Let v be a vertex of tenacity t. We will say that vertex u is $pred_t$ of v if either:

(i). u is a predecessor of v and $tenacity(u) \geq t$, or

(ii). there is a predecessor, u' of v, such that

$tenacity(u') < t$, and

$u = base_{\geq t}(u')$.

Define $pred^*_t$ to be the reflexive, transitive closure of the relation $pred$.

Remark: If v is outer, then only the matched neighbour of v is $pred_t$ of v. Thus (ii) applies only for inner vertices. In this case, there is an odd-alternating path from u to v of length $minlevel(v) - minlevel(u)$ whose internal vertices are all in the blossom $B_{u,t-2}$ (because a $minlevel(v)$ path consists of an $evenlevel(u')$ path concatenated with the edge (u',u)).

Theorem 7(b): Let (x,x') be the unique bridge of tenacity t on p. If $tenacity(x)<t$ then let $y_o = base_{\geq t}(x)$, otherwise let $y_o = x$. Similarly, if $tenacity(x')<t$ then let $z_o = base_{\geq t}(x')$, otherwise let $z_o = x'$. Then there is a set of distinct vertices $y_1 \cdots y_k, z_1 \cdots z_l$ such that

(i). y_{i+1} is $pred_t$ of y_i, for $0 \leq i < k$, and b is $pred_t$ of y_k.

(ii). z_{i+1} is $pred_t$ of z_i, for $0 \leq i < l$, and v is $pred_t$ of z_l.

(c): If (x,x') is matched then $oddlevel(x) = oddlevel(x') = (t-1)/2$. If (x,x') is unmatched then either $evenlevel(x) \leq (t-1)/2$ or $tenacity(x) < t$, and either $evenlevel(x') \leq (t-1)/2$ or $tenacity(x') < t$.

The following lemma will be used in the proof of Theorem 9.

Lemma 14: Let v_o be a vertex of tenacity t, and $v_1 \cdots v_k$ be distinct vertices such that v_i is $pred_t$ of v_{i-1} for $1 \leq i \leq k$. If $v_1 \cdots v_{k-1}$ are of tenacity t and $tenacity(v_k) > t$ then $v_k = base(v_o)$.

Proof: By the remark following the definition of $pred_t$, there is a path p from v_k to v_o of length $minlevel(v_o) - evenlevel(v_k)$. By the definition of $pred_t$, p is part of a $minlevel(v_o)$ path. Now, since all vertices on p, other than v_k, have tenacity $\leq t$, it follows that $base(v_o) = v_k$. □

Remark: Let $G(V,E)$ be a graph and M be a matching in it. Construct a new graph $G'(V',E')$ as follows: on each unmatched vertex of G add a new matched edge, and connect the other endpoints of these edges via unmatched edges to a single new unmatched vertex f. Now, there is an obvious one-to-one correspondence between $evenlevel(v)$ and $oddlevel(v)$ path in G and G', for $v \in V$: simply remove the first two edges of a path in G' to obtain the path in G. (Notice that the first two edges are props, since they give minlevels.) So, the evenlevel and oddlevel increases by 2 and the tenacity by 4 in going from G to G'. Therefore, Theorems 1 and 2 are true for G as well. Henceforth, let m denote the length of a minimum length augmenting path in G. For $v \in V$ of finite tenacity, let us say that *the base of v is defined* if $base(v) \neq f$ in G'. Notice that following is an alternative characterization of such vertices: on any $evenlevel(v)$ or $oddlevel(v)$ path in G, there is a vertex of $tenacity > tenacity(v)$. (Clearly, if $tenacity(v) < m$, the base of v is defined.) For such vertices, Theorems 3 to 6 hold in G as well because of the above-stated one-to-one correspondence. Finally, Theorem 7 holds for all vertices, v, in G because the first two edges on any $evenlevel(v)$ or $oddlevel(v)$ path in G' are props.

A description of the algorithm of [MV] appears in [PL]; however, the proof offered there is not complete. The structural development given above seems essential for giving a formal proof of correctness.

9. Procedure MIN, and the Role of Anamolies

Procedure MIN finds the minlevels of vertices, and it also determines whether an edge is a prop or a bridge. MIN examines an edge (u,v) at most once (MIN marks edges that it examines 'used' so they don't get examined again). If (u,v) is unmatched (matched), MIN examines this edge while searching from the endpoint having smaller evenlevel (oddlevel); ties are broken arbitrarily. Let us assume that (u,v) is examined from v. If at this stage MIN gives u its minlevel then (u,v) is a prop, otherwise it is a bridge.

If the search level i is even, MIN examines unmatched edges incident at vertices having evenlevel i. Suppose $evenlevel(v) = i$, (u,v) is unmatched and has not yet been examined by MIN. The following cases arise (we will assume that at the beginning of the phase, the evenlevels and oddlevels of all vertices are initialized to ∞):

(i). $minlevel(u) = \infty$: $minlevel(u)$ is set to $i+1$, v is inserted in the set of predecessors of u, and (u,v) is marked a prop.

(ii). $minlevel(u) = i+1$: v is inserted in the predecessor list of u, and (u,v) is marked a prop.

(iii). $minlevel(u) \leq i$ and $evenlevel(u)$ is finite: (u,v) is marked a bridge and is inserted in the set of bridges of tenacity $(evenlevel(u) + evenlevel(v) + 1)$. (See figure 11.)

(iv). $minlevel(v) \leq i$ and $evenlevel(u) = \infty$: v is inserted in the set of anamolies of u, and (u,v) is marked a bridge. (We will give a precise definition of anamolies below and will explain their significance. See figure 12 for an example.)

If the search level i is odd, MIN examines matched edges incident at vertices having oddlevel i. Suppose $oddlevel(v) = i$, and matched edge (u,v) has not yet been examined by MIN. The following cases arise:

(i). $minlevel(u) = \infty$: $minlevel(u)$ is set to $i+1$, v is inserted in the set of predecessors of u, and (u,v) is marked a prop.

(ii). $minlevel(u)$ is finite: in this case, $oddlevel(u) = i$. Edge (u,v) is marked a bridge and is inserted in the set of bridges of tenacity $(oddlevel(u) + oddlevel(v) + 1)$.

Definition: The following definition follows from case (iv). We will say that v *is an anamoly of* u if (u,v) is an unmatched edge, u is inner, and $oddlevel(u) < evenlevel(v) \leq (tenacity(u) - 1)/2$. In this case (u,v) is called an *anamoly*.

>From the above definition it is clear that an anamoly is a bridge, and that $tenacity(u,v) > tenacity(u)$. Notice that MIN is able to determine the tenacity of all bridges other than anamolies. In case (iv), $tenacity(u,v)$ cannot be determined since $evenlevel(u)$ is not yet found. At search level $(tenacity(u) - 1)/2$, MAX will find $evenlevel(u)$ and will determine the tenacity of all anamolies of u, including (u,v). This is well in time for calling DDFS with bridge (u,v), since $tenacity(u,v) > tenacity(u)$.

Notice that anamolies were not mentioned in the structure developed in Theorems 1 to 7. Anamolies are an algorithmic convenience, and the above-stated manner of handling them leads to the synchronization mentioned in Section 2.

10. Double Depth First Search

Procedure MAX uses a new graph searching algorithm: double depth first search (DDFS). We will first describe and prove correctness of this algorithm in the following simple setting:

Definition: A directed graph $H(S,T)$ with distinguished vertices $a,b \in S$ is said to be *layered* if S is partitioned into sets S_n , S_{n-1} , $\cdots$ S_o, called *layers*, such that:

(i). every edge goes from a higher numbered layer to a lower numbered layer (not necessarily consecutive), and

(ii). Vertices in S_n (S_o) have positive outdegree (indegree), and those in $S_{n-1} \cup \cdots \cup S_1$ have positive indegree as well as outdegree.

If $u \in S_k$, then $level(u) = k$. The distinguished vertices a and b can be at any levels. Vertex $s \in S_l$ is said to be a *bottleneck* if for every vertex u having level $\geq l$, every path from a to u and from b to u contains s, and moreover s is the highest leveled such vertex. We will assume that (u,v) represents the directed edge from u to v.

The problem is to determine if there is a bottleneck. Also (we will assume that a and b are initially marked 'L' and 'R' respectively):

(i). if there is a bottleneck, say $s \in S_l$, to find. Furthermore, to mark 'L' or 'R' (corresponding to left and right) all vertices having level $>l$ which are reachable from a or b. If such a vertex, u, is labeled L (R)

then there is a path from $a(b)$ to u consisting of L (R) marked vertices. There are also two paths, one from a to s and other other from b to s consisting of L and R marked vertices respectively.

(ii). if not, to find vertices c and $d \in S_o$, and vertex disjoint paths from a to c and b and d.

DDFS accomplishes this task in linear (i.e. $O(|T|)$) time. It simultaneously grows two vertex disjoint DFS trees T_L and T_R rooted at a and b, and consisting of L and R marked vertices respectively. DDFS maintains two centers of activity, c_L and c_R which start at a and b respectived. As in the usual DFS, if a center of activity moves from u to v, then $p(v) = u$, i.e. u is made the *parent* of v. Also, if a center of activity is at u and all outgoing edges from u have been examined, then the center of activity moves to $p(u)$; this important step is called *backtracking*. A pidgin Algol description of the procedure appears on the next page; for the sake of visual clarity we have used indentation to demarcate statements.

We now describe how the two DFS's are coordinated; we will first present and prove a quadratic time version of DDFS, and later make it linear time. If c_L and c_R are at different levels then the higher one grows its tree, and otherwise c_L grows its tree; the latter choice is arbitrary. The reflexive transitive closure of the relation parent is called *ancestor*. The ancestors of c_L (c_R) give a directed path from a to c_L (b to c_R) consisting of L (R) marked vertices; these vertices will be called the c_R) vertices. The most important step is dealing with the situation that c_L and c_R meet at a vertex w. In this case, first c_R attempts (another arbitrary choice) to find a new path from a right active vertex to a vertex having $level \leq level(c_L)$, by backtracking and finding new outgoing edges from right active vertices in the usual DFS manner. (Clearly, the path found will start at the lowest possible right active vertex.) If c_R succeeds, then w is included in T_L (i.e. marked 'L'), and the search proceeds. Otherwise, c_L attempts to find a path from a left active vertex to a vertex having $level \leq level(c_R)$. If c_L succeeds, then w is included in T_R (i.e. marked 'R'), and the search proceeds. If c_L also fails, then w is the bottleneck, and DDFS halts. In this case, w is not included in T_L or T_R. In the first two cases, $p(w)$ is appropriately set, depending on whether w is included in T_L or T_R. Finally, DDFS halts if c_L and c_R are both at distinct level 0 vertices.

It is easy to check that DDFS maintains the following invariant: any vertex that is visited (i.e. in $T_L \cup T_R$) but not active has been backtracked from. (Notice that there may be right active vertices that have already been backtraced from; however, every left active vertex has not been backtracked from.) It follows, that if DDFS halts at vertex s, then every vertex visited, other than s, is backtracked from. Using a straightforward proof by contradiction, one can now show that s is indeed the bottleneck, and furthermore, DDFS would have visited every vertex reachable from a or b and having $level > level(s)$. The remaining requirements follow from the fact that at any point in the algorithm, there is a path from a to c_L (b to c_r) consisting of L (R) marked vertices.

The above-stated algorithm follows any outgoing edge at most once. However, notice that c_R may backtrack from a vertex several times. (Though c_L backtracks from a vertex at most once because any left active vertex is not yet backtraced from.) This leads to an $O(|S|^2)$ running time. The algorithm is made more efficient as follows: initially, a *barrier* is placed at b. When c_L and c_R meet at a vertex, say w, c_R backtracks only up to the barrier. If it fails to find an alternative path, it includes w in T_R and it moves the barrier to w. The right active vertices now include the ancestors of c_R from c_R to the barrier only. The above-stated invariant is still maintained; in addition we have that any active vertex (left or right) is not yet backtraced from. This yields the following:

Theorem 8: DDFS accomplishes the above-stated tasks in $O(|T|)$ time. More precisely, if it ends with a bottleneck s, then the time taken is $O(|T'|)$, where T' is the set consisting of edges which are on a path from a or b to v.

11. Using DDFS to Find the Support of a Bridge, and Procedure MAX

We will first show how DDFS can be used to find the support of a bridge in an idealized setting. Let (u,v) be a bridge of tenacity $t \leq m$.

Let

$$u_o = \begin{cases} u & \text{if } tenacity(u) = t \\ base_{\geq t}(u) & o.w. \end{cases}$$

$$v_o = \begin{cases} v & \text{if } tenacity(v) = t \\ base_{\geq t}(v) & o.w. \end{cases}$$

This is illustrated in figure 11. Let $H(S,T)$ be the directed graph consisting of vertices $S = \{w \in V \mid w$ is $pred^*_t$ of u_o or $v_o\}$, and edges $T = \{(w,w') \mid w,w' \in S$, and w' is $pred_t$ of $w\}$. Partition vertices in S into layers according to their minlevels, thereby obtaining a layered graph $H(S,T)$.

Proposition 1: Suppose DDFS is run on graph $H(S,T)$ with distinguished vertices u_o and v_o. Then:

(i). if DDFS terminates with bottleneck s, then the set of vertices visited by DDFS, other than s, constituted support (u,v).

(ii). if DDFs terminates at two unmatched vertices f and f', then there is a minimum length augmenting path from f to f' containing (u,v). In this case $tenacity(u,v) = m$.

Proof: (i). Suppose $w \in$ support (u,v). by Theorem 7(b), $w \in S$. If the base of w is defined then let $b = base(w)$, otherwise let b be any unmatched vertex at which a $minlevel(w)$ path starts. By Theorem 7(b), $b \in S$, and there are two disjoint paths in $H(S,T)$, one from u_o (say) to w and the other from v_o to b. Since $minlevel(w) > minlevel(b)$, w is above the bottleneck and will be visited.

For the other direction, first note that s must be outer, since inner vertices in S have only one incoming edge. Suppose $w \neq s$ is visited by DDFS; assume w.l.o.g. that w is outer and is marked L, and that u_o and v_o are marked L and R respectively. Then there is an L marked path, p_1, from u_o to w and an R marked path, p_2, from v_o to s in $H(S,T)$.

Let $x,y \in S$, with x $pred_t$ of y. If x is a predecessor of y then (y,x) is an edge in G. Otherwise, by the remark following Theorem 7(a), there is a path from x to y of length $minlevel(y) - minlevel(x)$ in G. Using this fact (and since the paths mentioned above will be in distinct blossoms), it is easy to see that corresponding to p_1 and p_2 there are disjoint paths in G: p'_1 from w to u_o of length $minlevel(u_o) - minlevel(w)$, and p'_2 from s to v_o of length $minlevel(v_o) - minlevel(s)$. Let p'_3 be an $evenlevel(s)$ path. Now, p'_3, p'_2 and (u,v), together with an $evenlevel(u,u_o)$ path (if $u_o \neq u$), and $evenlevel(v,v_o)$ path (if $v_o \neq v$) gives a $maxlevel(u_o)$ path. This path concatenated with p'_1 gives an $oddlevel(w)$ path of length $t - minlevel(w)$. This proves that $tenacity(w) = t$ and $w \in$ support (u,v).

(ii). There are two disjoint paths, one from u_o to f and the other from v_o to f', in $H(S,T)$. As in (i), these correspond to disjoint paths in G which yield an augmenting path from f to f'. □

Remark: The bottleneck s found in case (i) will be the base of a blossom iff $tenacity(s) > tenacity(u,v)$. Figure 13(a) shows an example in which $tenacity(s_1) = tenacity(u_1, v_1)$.

Let the search level be i. MAX imposes an arbitrary ordering on the bridges of tenacity $2i+1$, say $g_1, g_2, \cdots g_k$, and calls DDFS with the bridges in this order. For the purposes of efficiency, the working of DDFS is different in two ways from the above-stated idealized setting. We explain the differences below.

Firstly, a vertex may be in the support of more than one bridge. For example vertex w in figure 13(a) is in the support of (u_1, v_1) and (u_2, v_2). Now, w will be visited by MAX only once and will be assigned to one bridge: to (u_1, v_1) if DDFS is called with (u_1, v_1) before (u_2, v_2), and to (u_2, v_2) otherwise. Define $petal(g_i) = support(g_i) - \bigcup_{j<i} support(g_i)$ DDFS finds the petals of $g_1 \cdots g_k$, rather than their supports. Notice that unlike blossom and support which are graph-theoretically defined, petals are algorithmically defined since they depend on the ordering imposed on the bridges. Suppose (u_1, v_1) is processed before (u_2, v_2) in figure 13(a). The first DDFS ends with bottleneck s_1 (notice that

$tenacity(s_1) = tenacity(u_1, v_1)$), and the second with s_2. Define the *bud* of a petal to be its bottleneck. Thus $bud(petal(u_1, v_1)) = s_1$ and $bud(petal(u_2, v_2)) = s_2$. Also, if $w \in petal(u,v)$, define $bud(w)$ to be $bud(petal(u,v))$. If the newly found petal is non-empty as an implementational convenience, DDFS creates a new node; all the vertices of the petal point to the node, and the node points to the bud. This is illustrated in figure 13(b) and 14. These figures also show the 'L' and 'R' marks left by DDFS.

The second difference is that the graph $H(S,T)$ is not constructed explicitly; DDFS in run on graph $G(V,E)$ itself as follows. Suppose the center of activity is at v and u is a predecessor of v. If u is not in any petal, then the center of activity moves to u. Otherwise (in this case $tenacity(u) \leq tenacity(v)$), the center of activity moves to $bud^*(u)$. The function $bud^*(u)$ is defined below:

function $bud^*(u)$
  if u is not in a petal then return u
    else return $bud^*(bud(u))$.
end;

One more point needs to be mentioned. Suppose DDFS is called with bridge (u,v). If u is in a petal, then the left center of activity starts from $bud^*(u)$; similarly for v.

Having found the vertices of tenacity $2i+1$, MAX determines their maxlevels. For each such vertex v, and for each anamoly u of v, MAX also determines the tenacity of bridge (u,v).

12. Proof of Correctness of MIN and MAX

In Theorem 9, we will show why the synchronization works, and we will establish several properties of petals and buds in order to prove the correctness of MIN and MAX.

Theorem 9: Let m be the length of a minimum length augmenting path in G. Then MIN and MAX correctly find the minlevel of all vertices having $minlevel \leq m$, and the maxlevel of all vertices having $tenacity \leq m$.

Proof: By induction on i, the search level, we will prove the following stronger statement:

Induction Hypothesis: At the end of search level i (assume $t = 2i+1$):

1). all vertices v s.t. $minlevel(v) \leq i+1$ get their correct minlevel; the remaining vertices have their minlevel set at ∞.

2). all vertices v s.t. $tenacity(v) \leq t$ get their correct maxlevel, and are assigned a petal. The petal of vertices of higher tenacity is still undefined, and their maxlevel is set at ∞.

3). for any vertex v s.t. $tenacity(v) \leq t$, $bud^*(v) = base_{>t}(v)$.

The hypothesis is clearly true for search level. 0. Assuming its truth for search levels $<i$, we prove it true for search level i.

(1). At the beginning of search level i the following holds: if i is even (odd), every vertex u having an even-level (oddlevel) of i would have gotten it. (If $minlevel(u) = i$, this follows from (1). If $maxlevel(u) = i$, then $tenacity(u) \leq 2i-1$, and the assertion follows from (2)). Suppose $minlevel(v) = i+1$, and let (u,v) be the last edge on a $minlevel(v)$ path. By the above-stated fact, MIN will find v while searching from u. In this manner, MIN finds all the predecessors of v, and is correctly able to distinguish props from bridges.

(2.) This involves proving statements (1) and (2) made at the end of section 2; (3) is proved in Theorem 7. Since MIN correctly distinguishes between props and bridges, to prove the first statement, we only need to show that the tenacity of bridges is determined 'well in time'; this is done below in Lemma 15 for bridges having non-empty support.

Let $g_1, g_2 \cdots g_k$ be an arbitrary ordering imposed by MAX on the bridges of tenacity $t = 2i+1$. By induction on j, we will prove that when DDFS is called g_j it finds $petal(g_j)$, thereby proving the second statement.

The induction basis follows from Proposition 1 and induction hypothesis (3), because DDFS is essentially being run on the associated graph $H(S,T)$. We prove the induction step below.

Consider the situation when DDFS is called with bridge g_j. We make the following observations: Suppose vertex v is in a petal, and suppose u, a predecessor of v, is not in a petal. Then clearly $u = bud^*(v)$. Secondly, if vertex v is in a petal and $tenacity(v) = t$, then $bud^*(v)$ is $pred^*_t$ of v. From the first observation it follows that if DDFS arrives at v, it is sufficient to continue search from $bud^*(v)$ only. From the second observation it follows that DDFS visits all vertices in $support(g_j) - \bigcup_{l<j} petal(g_l)$. But by the induction hypothesis, this set is $support(g_j) - \bigcup_{l<j} support(g_l) = petal(g_j)$.

(3.) Let v be a vertex of tenacity t, and $b = bud^*(v)$ at the end of search level i. Clearly, b is $pred^*_t$ of v. Since b is not in the support of any bridge of tenacity t, $tenacity(b) > t$. Now, by Lemma 14, $base(v) = b$. Next suppose $tenacity(v) < t$. Let b' be $bud^*(v)$ at the end of search level $i-1$. By induction hypothesis (3), $b' = base_{\geq t}(v)$. If $tenacity(b') > t$, b' will be $bud^*(v)$ at the end of search level i also. If $tenacity(b') = t$, $b = bud^*(b')$ will be $bud^*(v)$ at the end of search level i. By the above-stated argument, $b = base_{>t}(v)$. □

Remark: Let b be an outer vertex of tenacity $>t$, and let $B_{b,t}$ be the blossom having base b and tenacity t. Then, hypothesis (3) implies that at the end of search level i, the set $\{v \in V \mid bud^*(v) = b\} = B_{b,t}$.

Lemma 15: Let (u,v) be a bridge of tenacity $2i+1$ having non-empty support. Assuming the induction hypothesis of Theorem 9, the algorithm will determine the tenacity of (u,v) by the end of execution of MIN during search level i.

Proof: Suppose (u,v) is matched. By Theorem 7(c), $oddlevel(u) = oddlevel(v) = i$. Therefore, during the execution of MIN during search level i, (u,v) will be examined and its tenacity will be assertained.

Next suppose (u,v) is unmatched. *W.l.o.g.* assume $evenlevel(u) \leq evenlevel(v)$. Since $tenacity(u,v) = 2i+1$, $evenlevel(u) \leq i$. Edge (u,v) will be examined from u at search level $evenlevel(u)$. If $evenlevel(v)$ is determined at this stage, $tenacity(u,v)$. is ascertained. Otherwise, $evenlevel(v)$ must be $>i$. If so, by Theorem 7(c), $tenacity(v) < 2i+1$. This implies $evenlevel(v) > oddlevel(v)$, i.e. v is inner. Since u is not a predecessor of v, $evenlevel(u)+1 > oddlevel(v)$. So, while searching from u at search level $evenlevel(u)$, MIN will make u an anamoly of v. Now, at search level $(tenacity(v)-1)/2$, i.e. before search level i, $evenlevel(v)$ will be given, and $tenacity(u,v)$ will be ascertained. □

Remark: A matched bridge (u,v) has non-empty support, since its support contains u and v. One can extend Lemma 15 to unmatched bridges having empty support using the following easily proven assertions (assume (u,v) is an unmatched edge):

a). if u is not a predecessor of v and v is inner, then $tenacity(v) < tenacity(u,v)$.

b). if v is outer, then $evenlevel(v) \leq (tenacity(u,v)-1)/2$.

Figure 15 illustrates the importance of this synchronization. Notice that $tenacity(z,z') = 25$ is ascertained at search level 10 while processing bridge (u,u'). So, why not call DDFS with (z,z') at search level 10? If this is done, then the vertices visited may not be in $support(z,z')$, and so DDFS will not be able to ascertain their tenacity. For example, v will be visited, though $v \in support(w,w')$ and has tenacity 23.

13. Finding Minimum Length Augmenting Paths

If the current matching, M, is not maximum then at search level $(m-1)/2$ DDFS will eventually be called with a bridge that is on a minimum length augmenting path, and will end up at two unmatched vertices; m denotes the length of a minimum length augmenting path w.r.t. M. At this point procedure FINDPATH is

invoked to find such a path between the two vertices. On the other hand, if the current phase executes for $(|V|-1)/2$ search levels without finding an augmenting path, matching M is maximum, and the algorithm halts.

DDFS marks petals appropriately so that FINDPATH may find a path through the petals efficiently. Suppose DDFS is called with bridge (u,v). Let $u_o = bud^*(u)$ and $v_o = bud^*(v)$, and assume that the left and right centers of activity start at u_o and v_o respectively. After the new petal P is formed, the following are set (assuming P is non-empty): L-peak $(P) = u$ and R-peak $(P) = v$. As shown in figures 13(b) and 14, the new petal-node points to L-peak and R-peak.

Furthermore, if $u_o \neq u$, DDFS sets $exit\text{-}bud(u,v) = u_o$. Similarly, if $v_o \neq v$, $exit\text{-}bud(v,u) = v_o$. Suppose DDFS visits vertex x, y is a predecessor of x, y is already in a petal, and $y_o = bud^*(y)$. Then, DDFS visits y_o, and sets $exit\text{-}bud(y,x) = y_o$. In this case, y_o is $pred^*_t$ of x, where $t = tenacity(u,v) = tenacity(x)$. This is illustrated in figure 14.

Let us say that u is $bud^+(v)$ if $u = bud^*(v)$ at some point during the execution of the current phase. FINDPATH is called with two vertices, say x and y, as parameters, where y is $bud^+(x)$. It returns an even-alternating path from y to x of length $evenlevel(x) - evenlevel(y)$. Assume x is in petal P, is marked R and has tenacity t. FINDPATH is a recursive procedure; it first finds the 'shell' of the required path. First suppose x is an outer vertex. FINDPATH simply follows predecessors till it gets to $bud(x)$: suppose it is at vertex $z \in P$ and w is a predecessor of z. If $w \in P$, FINDPATH adds w to the shell. If $w \notin P$, it finds $exit\text{-}bud(w,z)$, say w', and adds w and w' to the shell. It then continues search from w'. If $bud(x) \neq y$, FINDPATH continues the above process; notice that $bud(x)$ is outer and y is $bud^+(bud(x))$. Finally, FINDPATH fills in the 'gaps' in the shell (such as (w,w')) by recursively calling FINDPATH (e. g. on parameters w,w'). When all the recursive calls are complete, the path from y to x has been found. For example, the call FINDPATH(o,q) in figure 16 results in the shell $m\ n\ o\ q$. The recursive call to fill the gap is FINDPATH(o,q).

If x is an inner vertex, the process is more involved since the path from y to x will use the bridge of P. FINDPATH first finds the left and right peaks of p, say u and v. Suppose $exit\text{-}bud(u,v) = u_o$ and $exit\text{-}bud(v,u) = v_o$. FINDPATH now grows a DFS tree rooted at v_o and consisting only of the R marked vertices of P till it finds x. This yields a shell from x to v_o. It then grows another DFS tree rooted at u_o and consisting only of the L marked vertices of P till it finds $bud(x)$. These shells are filled in through recursive calls. Also paths are found from u_o to u and v_o to v. These concatenated with (u,v) give an even-alternating path from $bud(x)$ to x. The path from y to $bud(x)$ is found as stated above. For example, the call FINDPATH(j,f_1) in figure 16 results in the two shells $f\ e\ d\ c\ b$ and $g\ h\ i\ j$; these have no gaps. These are concatenated with the path $b\ a\ f_1$ from b to f_1.

Suppose DDFS is called the bridge (u,v) at search level $(m-1)/2$, and ends at unmatched vertices x and y. Suppose $u_o = bud^*(u)$ and $v_o = bud^*(v)$. Then, an augmenting path from x to y is found by the following: FINDPATH (u_o, x) ~ $FINDPATH(u,u_o)$ ~ (u,v) ~ $(FINDPATH(v_o, v))^{-1}$ ~ $(FINDPATH(y,v_o))^{-1}$. In figure 16, the augmenting path is obtained by FINDPATH(u,f_1) ~ (u,v) ~ FINDPATH $(v,f_2)^{-1}$. This results in the recursive calls FINDPATH (j,f_1), FINDPATH (m,q) and FINDPATH (s,y).

After a path p is found, MAX invokes procedure TOPOLOGICAL ERASE. This procedure erases p and all vertices which cannot be in a minimum length augmenting path disjoint from p as follows: TOPOLOGICAL ERASE first erases all vertices of p and all edges incident at these vertices. Then, it successively removes any remaining vertices which have no predecessors left. After this, MAX continues processing the remaining bridges of tenacity m to find augmenting paths in the remaining graph. In this manner it finds a maximal set of disjoint minimum length augmenting paths.

14. Running Time of the Algorithm

Finally, we analyse the time taken by the algorithm to execute one phase. Since MIN examines each edge only once, it takes $O(|E|)$ time. A vertex belongs to at most one petal, and DDFS examines its predecessor edges once during the formation of this petal. However, DDFS also has to compute bud^* of vertices.

This can be accomplished in $O(|E| \propto(|E|, |V|))$ time using the set union algorithm of [Ta]; here $\propto$ is the inverse Ackerman function. By resorting to the RAM model of computation (in which operations on $O(logn)$ bit numbers are assumed to take unit time), Gabow and Tarjan [GT1] have given an incremental tree set union algorithm which gives a linear time implementation of *bud** on the RAM model. We leave the open problem of obtaining a linear time implementation of *bud** in the pointer machine model of computation (see [Kn] for a precise definition). One avenue for accomplishing this is to prove the claim in [MV] that because of the special structure of blossoms, if *bud** is implemented using only path compression, its cost can be charged to the edges and is linear.

Let us analyse the time taken by FINDPATH in a phase. Suppose vertex v which is in petal P is on a minimum length augmenting path p. Let $b = base_{\geq m}(v)$. Clearly, b is also on p. Let $B_{b,m}$ be the blossom having base b and tenacity m. Clearly $P \subseteq B_{b,m}$. Now, a minimum length alternating path from any unmatched vertex to a vertex in $B_{b,m}$ must use b. Therefore, the vertices in $B_{b,m}-p$ cannot be on a minimum length augmenting path disjoint from p, and will be deleted by TOPOLOGICAL ERASE. Therefore, FINDPATH does at most two DFS's in petal P.

Let (u,v) be a prop, with u predecessor of v. Define $petal(u,v)$ to be $petal(u)$, and define the size of a petal to be the number of edges in it. By the above-stated remarks, the work done by DDFS in a petal is linear in its size. Therefore, the total time taken by FINDPATH in a phase is $O(|E|)$. It is easy to see that TOPOLOGICAL ERASE also takes linear time. This proves that the running time of the algorithm on the RAM model is $O(\sqrt{|V|}\,|E|)$.

Acknowledgements: I wish to thank Silvio Micali for memorable times spent discovering this algorithm. Thanks to Richard Karp for sharing his perspective on graph algorithms through courses given at Berkeley, and for introducing me to matching theory. I have benefited greatly from Manuel Blum's style of seeking simple explainations for seemingly complex phenomena. Thanks Manuel, for your encouragement and support. A special thanks to Steve Mitchell and Umesh Vazirani for spending several days verifying the proofs and suggesting simplifications, and for offering valuable comments on the writeup. Thanks also to the numerous other people who saw parts of this proof at various points during its development.

REFERENCES

[Be] Berge, C., *Two theorems in graph theory*, Proc. Natl, Acad. Sci., 43 (1957), pp. 842-844.

[Ed] Edmonds, J., *Paths, trees, and flowers*, Canad. J. Math., 17 (1965), pp. 449-467.

[Ga] Gabow H. N., *An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs*, JACM, vol. 23, No. 2, 221-234 (1976).

[GT1] Gabow, H. N., and Tarjan, R. E., *A linear-time algorithm for a special case of disjoint set union*, J. Comput. System Sci. 30, 209-221. (1985)

[GT2] Gabow, H. N., and Tarjan, R. E., *Faster scaling algorithms for general graph matching problems*, technical report CU-CS-432-89, University of Colorado at Boulder.

[HK] Hopcroft, J. E., and Karp, R. M., *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225-231.

[KM] Kameda, T. and Munro, I., *An O(|V|. |E|) Algorithm for Maximum Matching of Graphs* Computing, Vol. 12, 91-98 (1976).

[Kn] Knuth, D. E., *The Art of Computer Programming, Vol.* 1: *Fundamental Algorithms*, 2nd ed., Addison -Wesley, Reading, MA, 1973.

[La] Lawler, E. G., *Combinatorial Optimization Theory*, Holt, Rinehart and Winston, 217-239 (1976).

[LP] Lovasz, L., and Plummer, M., *Matching Theory*, Academic Press, Budapest, Hungary, (1986).

[MV] Micali, S., and Vazirani, V. V., *An $O(\sqrt{|V|}\,|E|)$ algorithm for finding maximum matching in general graphs*, in Proc. 21st Annual IEEE Symposium on Foundations of Computer Science,

1980, pp. 17-27.

[Ta] Tarjan, R. E., *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comput. Mach., 22 (1975), pp. 215-225.

[PL] Peterson, P.A., and Loui, C. *The general graph matching algorithm of Micali and Vazirani*, Algorithmica 3 (1988) pp. 511-533.

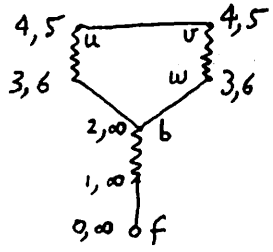


Figure 1.

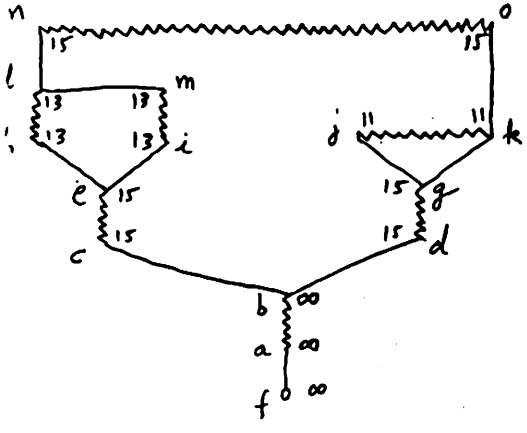


Figure 2.

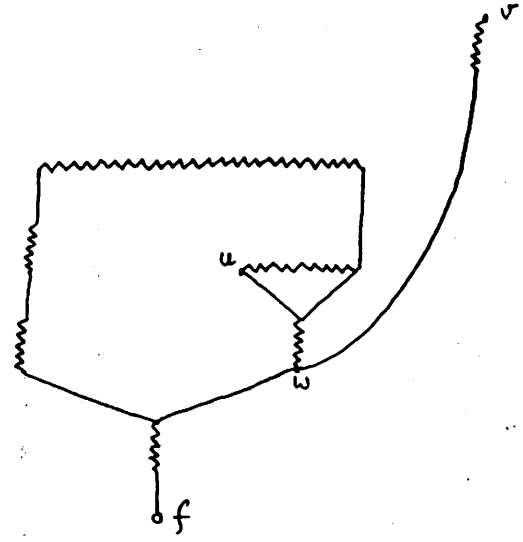


Figure 3.

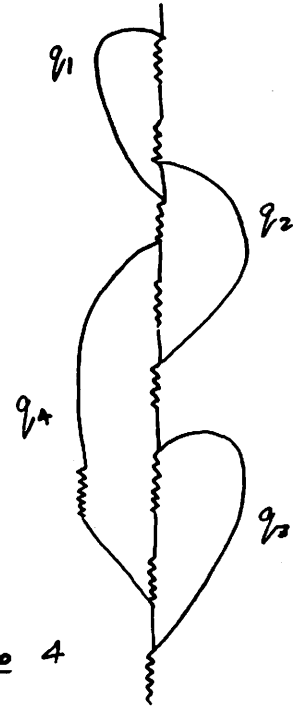


Figure 4

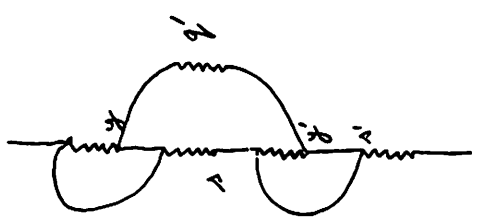


Figure 5.

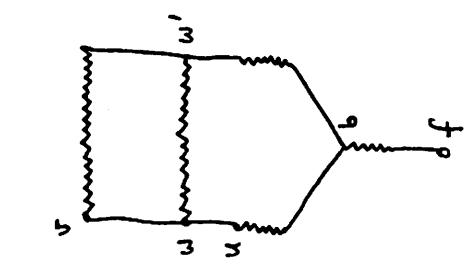


Figure 6.

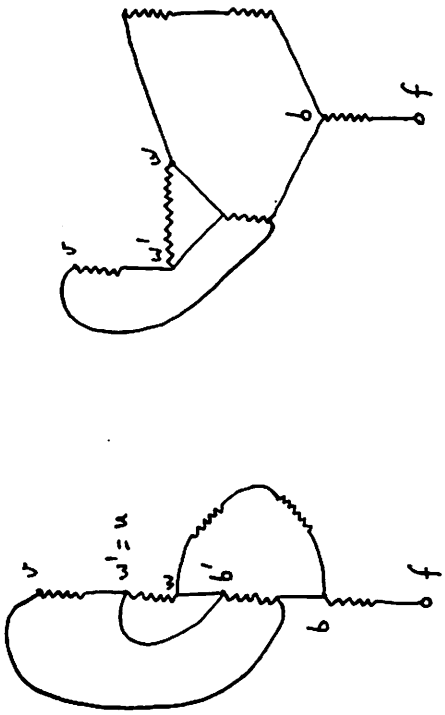


Figure 7.

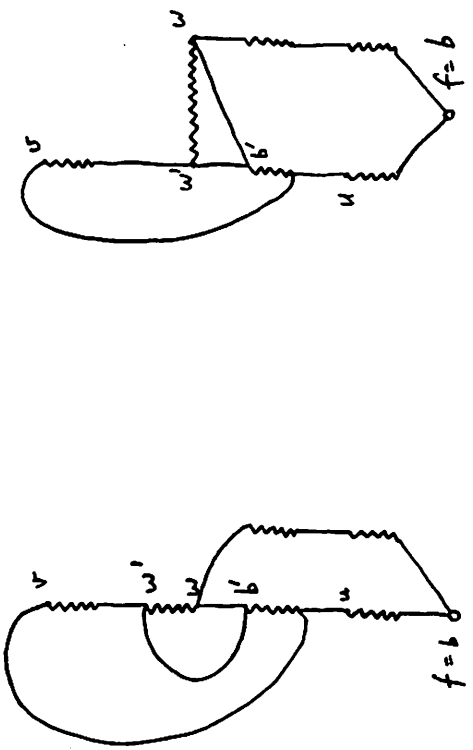
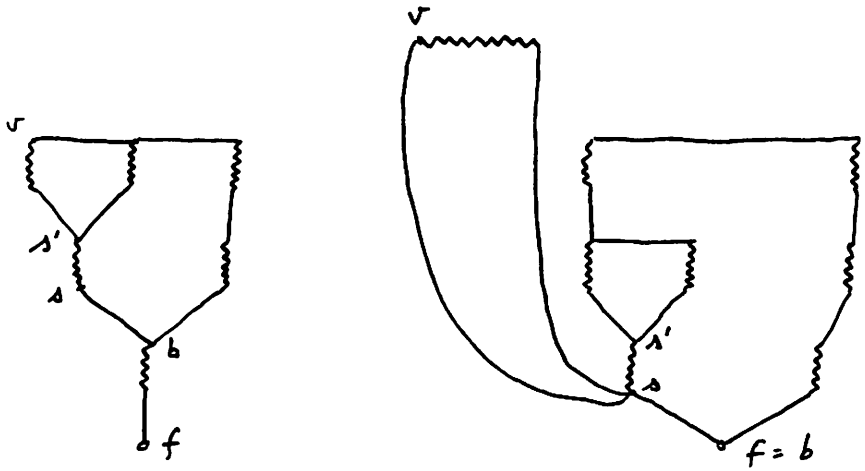


Figure 8.



(a) (b)

Figure 9

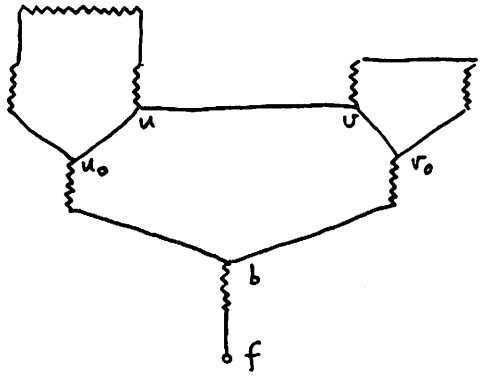


Figure 11.

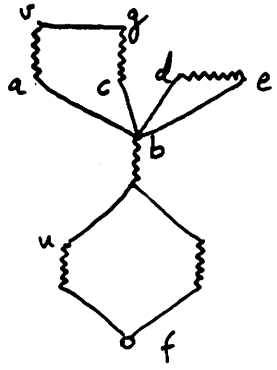


Figure 10

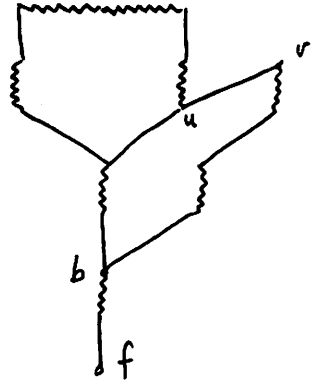
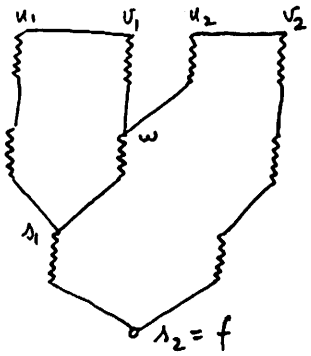
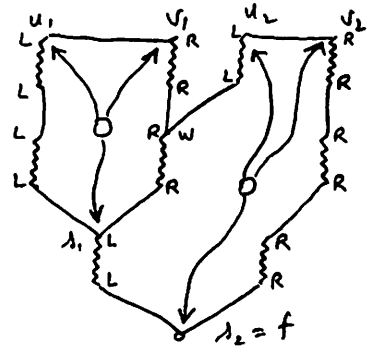


Figure 12.



(a)



(b)

Figure 13

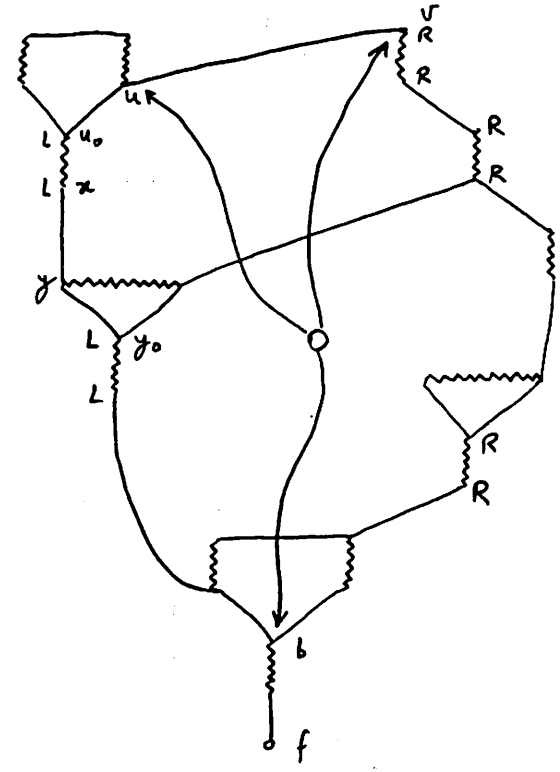


Figure 14.

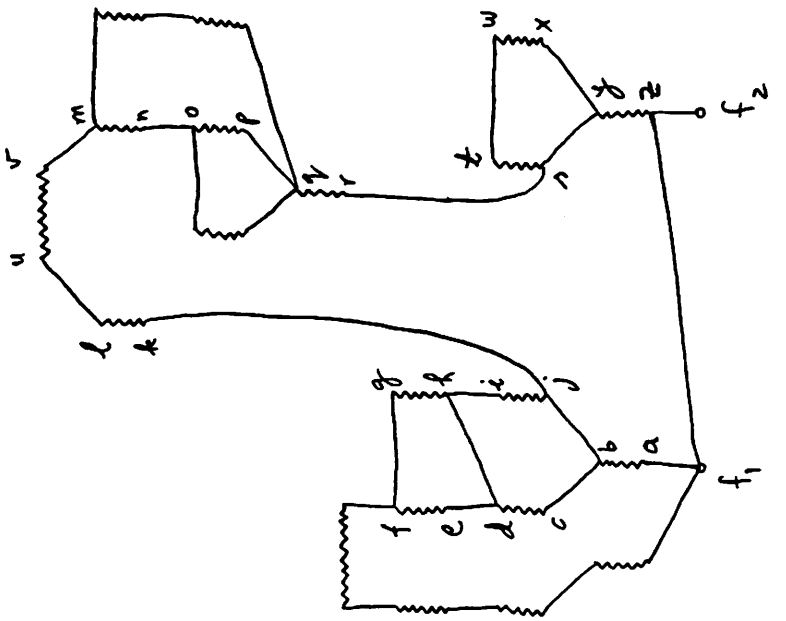


Figure 16.

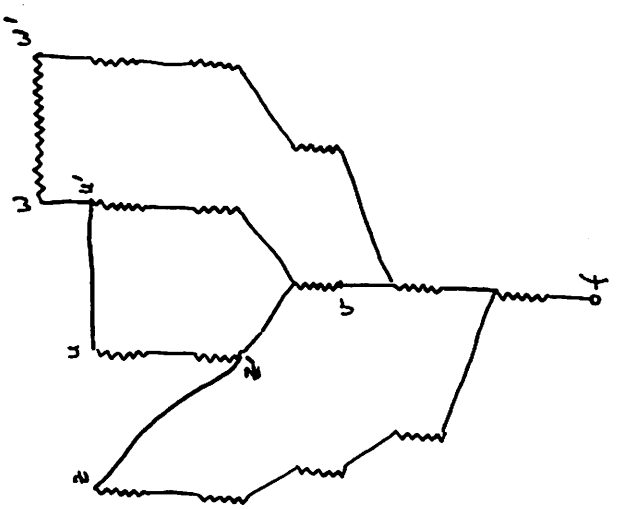


Figure 15.