imizes

$$\sum_{i \text{ before } j} \bar{c}_{ij}.$$

It can be formulated as an integer programming problem by introducing binary variables x_{ij} for each ordered pair i, j. The value of x_{ij} is one if i is before j in the ordering and zero otherwise. A necessary and sufficient condition for x to be the incidence vector of an ordering is that

$$x_{ij} + x_{jk} + x_{ki} \leq 2$$

for each ordered subset $\{i, j, k\}$ of V. Thus, the linear ordering problem can be stated as

$$\begin{array}{lrcll} \max & \bar{c}^T x & & & \\ \text{subject to} & x_{ij} + x_{jk} + x_{ki} & \leq & 2 & \text{for all ordered sets } i, j, k \text{ in } V \\ & x_{ij} + x_{ji} & = & 1 & \text{for all pairs } i, j \text{ in } V \\ & x_{ij} & = & 0 \text{ or } 1 & \text{for all ordered pairs } i, j \text{ in } V. \end{array}$$

3. The Cutting Plane Algorithm. Like Grötschel et al. [4], we used the equalities $x_{ij} + x_{ji} = 1$ to eliminate half the variables, leaving only x_{ij} for $i < j$. If $i < j < k$ then we get two 3-dicycle inequalities:

$$\begin{array}{rcl} x_{ij} + x_{jk} - x_{ij} & \leq & 1 \\ -x_{ij} - x_{jk} + x_{ij} & \leq & 0. \end{array}$$

If t^1_{ijk} and t^2_{ijk} are the slack variables for these constraints, then we must have $0 \leq t^l_{ijk} \leq 2$. At any iteration, the current linear programming relaxation of the linear ordering problem will have the form

$$\begin{array}{lccccccccccc} \min & c^T x & & & & & & & & & & \\ \text{subject to} & Ax & + & t & & & = & b & & & & (LOLP) \\ & x & & & + & s & = & e & & & & \\ & & & t & + & w & = & 2e & & & & \\ & x & \geq 0, & t & \geq 0, & s & \geq 0, & w & \geq 0, & & & \end{array}$$

where e is a vector of ones of the appropriate dimension, $c_{ij} = \bar{c}_{ji} - \bar{c}_{ij}$ for each $i < j$, and $Ax + t = b$ represents the 3-dicycle inequalities in the current relaxation.

When we added 3-dicycle constraints, we ensured that the set of additional constraints contributed at most one extra nonzero to any column of A; we call such a set of constraints *arc-disjoint* constraints. We found that this considerably improved the performance of the algorithm because the work required to calculate a projection is greatly decreased. Because of this choice, many more relaxations are examined and more iterations are taken, but the cost of this is easily outweighed by the improvement in the time required each iteration.

When there are no 3-dicycle inequalities in $(LOLP)$, the solution is to take $x_{ij} = 1$ if $c_{ij} < 0$ and 0 otherwise. Our initial set of constraints was an arc-disjoint subset of the set of all constraints violated by this point. Our initial primal and dual points were selected as in [9]. We used routines from IBM's Extended Scientific Subroutine Library (ESSL) [6] to calculate the projections necessary in an interior point method. We regarded the linear ordering problem as solved when we found an ordering satisfying

$$\frac{\text{Value of ordering} - \text{dual value}}{\max\{1, \text{absolute value of dual value}\}} \leq 10^{-6}.$$

We attempted to round the current solution x to $(LOLP)$ to an ordering every time we searched for violated 3-dicycle inequalities.

It is possible to look for 3-dicycle constraints violated by the current primal solution x at every iteration; however, we found it more efficient to wait until the relative duality gap

$$\frac{\text{Duality gap}}{\max\{1, \text{absolute value of dual value}\}}$$

was smaller than some tolerance. This tolerance was dynamically altered depending upon the largest 3-dicycle constraint violation. Cutting planes were found by enumerating the values of all the 3-dicycle constraints, bucket sorting them, and then looking for an arc-disjoint subset in the buckets corresponding to the most violated constraints. If we found constraints we wished to add, we also attempted to drop constraints. We drop a constraint if the primal slack t^l_{ijk} is at least 0.4 and if the constraint was not added at the previous stage.

When cutting planes are added, the current dual point is still feasible so it is not updated. However, the primal solution x becomes infeasible. In principle, this is not a problem: the algorithm presented by Lustig et al. [9] is an infeasible interior point algorithm, so it can start from any point which strictly satisfies the nonnegativity constraints. However, usually many variables are close to their bounds and if we restart from such a point the algorithm will take several iterations moving towards the center of the polyhedron before making progress to decrease the objective function. One possible solution to this is to move variables with small values away from their bounds; unfortunately, we found that after such a procedure the algorithm tended to then need several iterations to regain feasibility before it again starts to make progress. Therefore, we introduced a vector x^{FEAS} which is always feasible in the problem $(LOLP)$ and satisfies the inequality constraints strictly; when constraints are added we set $x = x^{FEAS}$. Initially, x^{FEAS} is the vector of halves; we attempt to update it at every iteration by moving in the direction from x^{FEAS} to the current primal solution x — the calculation of the maximum possible step in this direction can be found by a minimum ratio test. We then found that when cutting planes are added, the only centering necessary was for numerical reasons — if any variable is within 10^{-8} of its bounds it is set to 10^{-8}.

4. Computational Results and Discussion. Table 1 contains the results of our algorithm on 42 real-world linear ordering problems. These runs were made on an IBM ES/9580 under the AIX/370 operating system. The code was written in Fortran and compiled with the FORTRANVS compiler. CPU times were measured with the CPUTIME subroutine of the Fortran run-time library. These times include the time to read in the problem.

All of the problems come from input-output tables dating from 1954 to 1975. Problems whose names start with t are 44 by 44 tables available from EUROSTAT in the European Community. The second and third characters in the name give the year of the table and the fourth character identifies the country: b is Belgium, d is (West) Germany, e is Spain, f is France, i is Italy, k is Denmark, l is Luxembourg, n is the Netherlands, r is Ireland, u is the United Kingdom, w is the compilation of the data for the six original members of the European Community, and x is the compilation of the data for the first nine members of the E.C. t70d11b is one of two tables available from EUROSTAT for Germany for the year 1970. There are two additional tables available from EUROSTAT: t59b11 and t70d11a: neither of these problems can be solved by

TABLE 4.1
Results on 42 input-output matrices

Problem	Initial m	Cuts added	Stages	Final m	Iterations	Time (seconds)
t59d11	67	340	14	318	85	17.47
t59f11	76	232	11	247	64	11.71
t59i11	80	224	15	258	77	13.42
t59n11	73	374	16	311	78	15.23
t65b11	93	358	20	342	113	23.65
t65d11	105	297	18	302	85	16.36
t65f11	95	353	25	359	124	24.36
t65i11	97	389	16	406	98	20.59
t65l11	77	174	10	192	56	9.50
t65n11	104	412	17	420	94	21.58
t65w11	101	324	14	340	69	13.48
t69r11	75	345	19	327	109	21.82
t70b11	86	275	12	286	80	15.40
t70d11b	132	363	11	404	66	14.38
t70f11	104	337	14	387	68	11.92
t70i11	85	426	19	414	111	24.02
t70k11	73	345	16	362	90	18.56
t70l11	75	264	16	239	69	12.33
t70n11	94	346	20	353	98	20.12
t70u11	63	173	11	185	56	6.47
t70w11	110	240	12	303	62	11.39
t70x11	105	282	13	325	66	10.01
t74d11	125	319	13	369	71	14.71
t75d11	124	351	14	398	72	14.92
t75e11	100	442	16	411	99	19.66
t75i11	88	418	17	428	102	22.28
t75k11	81	295	12	320	69	13.38
t75n11	86	352	18	345	101	21.48
t75u11	92	254	16	286	73	13.46
Mean	92	321	15	332	83	16.33
diw56n54	210	721	23	683	119	53.92
diw56n58	206	625	16	636	80	34.33
diw56n62	205	876	21	776	112	61.21
diw56n66	204	817	16	729	89	42.98
diw56n67	211	809	17	768	113	63.13
diw56n72	213	710	16	716	90	42.23
diw56r54	211	723	22	706	100	37.87
diw56r58	207	632	19	626	83	35.55
diw56r66	206	755	15	703	81	37.50
diw56r67	217	816	18	759	94	48.11
diw56r72	206	731	17	711	91	43.84
Mean	209	747	18	710	96	45.52
sbm1974	225	928	17	895	112	79.08
sbm1975	231	1034	19	965	114	92.97
Mean	228	981	18	930	113	86.03

A PRIMAL-DUAL INTERIOR POINT CUTTING PLANE METHOD FOR THE LINEAR ORDERING PROBLEM

JOHN E. MITCHELL* AND BRIAN BORCHERS†

Abstract. A cutting plane algorithm for solving the linear ordering problem is described. This algorithm uses the primal-dual interior point method to solve the linear programming relaxations which arise. The algorithm attempts to identify cutting planes early and it stores an old feasible point which is used to help recenter when cutting planes are added. Computational results are described for some real-world problems; the algorithm appears to be somewhat competitive with a simplex-based cutting plane algorithm.

1. Introduction. The linear ordering problem is an NP-hard combinatorial optimization problem which has many real-world applications, including the triangulation of input-output matrices in economics. The polyhedral structure of the problem has been investigated by Grötschel, Jünger, and Reinelt [4,7,11]. They described a simplex-based cutting plane algorithm for solving the linear ordering problem and were able to solve sets of European Community and (West) German input-output tables. They showed that almost all of these problems could be solved using just simple bounds and *3-dicycle* inequalities.

We used the primal-dual logarithmic barrier method as described by Lustig, Marsten and Shanno [9]. This is a barrier function method which works on the primal and dual linear programs simultaneously, applying Newton's method to the sum of the objective function and the barrier function. All parameters were as described in [9], except as detailed in Section 3.

It is necessary to be able to exploit a "warm start" in the interior point method if it is to be used efficiently in a cutting plane algorithm. Usually, the solution to one relaxation is on the boundary of the feasible region, and interior point methods generally do not perform well when started from close to the boundary. We used two strategies to overcome this problem. The first was to attempt to identify cutting planes early, so the iterates do not get too close to the boundary. The second was to store a strictly feasible interior point from which the algorithm was restarted when cutting planes were added. As can be seen from the computational results in section 4, the method appeared to be able to efficiently exploit warm starts.

Computational work on solving integer programming problems using interior point methods has been done by Borchers [2], Kaliski and Ye [8], Mitchell and Todd [10], and Resende and Veiga [12], among others. For interesting theoretical results, see the papers cited above and also Goffin and Vial [3] and den Hertog et al. [5], which discuss the related problem of using an interior point method in a column generation setting.

2. The Linear Ordering Problem. The linear ordering problem can be stated as follows:

> Given a set of objects V, and rewards $\bar{c}_{ij}$ and $\bar{c}_{ji}$ for each pair of objects i, j in V, find a complete ordering for the objects which max-

* Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180 USA – mitchj@rpi.edu. Research partially supported by ONR Grant number N00014–90–J–1714.
† Department of Mathematics, New Mexico Tech, Socorro, NM 87801 USA – borchers@nmt.edu

REFERENCES

[1] *AAAI-88 Workshop on Parallel Algorithms for Machine Intelligence*, St. Paul, Minneapolis, August 1988. Organizers: V. Kumar, L. Kanal, P.S. Gopalakrishnan; Sponsored by American Association for Artificial Intelligence.

[2] *IJCAI-89 Workshop on Parallel Algorithms for Machine Intelligence*, Detroit, Michigan, 20 August 1989. Organizers: V. Kumar, L. Kanal, P.S. Gopalakrishnan; Sponsored by American Association for Artificial Intelligence.

[3] S. Arvindam, Vipin Kumar, and V. Nageshwara Rao. Floorplan optimization on multiprocessors. In *Proceedings of the 1989 International Conference on Computer Design (ICCD-89)*, 1989. Also published as MCC Tech Report ACT-OODS-241-89.

[4] S. Arvindam, Vipin Kumar, and V. Nageshwara Rao. Efficient parallel algorithms for search problems: Applications in vlsi cad. In *Proceedings of the Frontiers 90 Conference on Massively Parallel Computation*, October 1990.

[5] S. Arvindam, Vipin Kumar, V. Nageshwara Rao, and Vineet Singh. Automatic test pattern generation on multiprocessors. *Parallel Computing*, 17, number 12:1323–1342, December 1991.

[6] J. Dongarra, A.H. Karp, K. Kennedy, and D. Kuck. 1989 gordon bell prize (special report). *IEEE Software*, May 1990.

[7] Ananth Grama, Vipin Kumar, and V. Nageshwara Rao. Experimental evaluation of load balancing techniques for the hypercube. In *Proceedings of the Parallel Computing 91 Conference*, 1991.

[8] George Karypis and Vipin Kumar. Unstructured Tree Search on SIMD Parallel Computers. Technical Report 92–21, Computer Science Department, University of Minnesota, 1992.

[9] V. Kumar, P.S. Gopalkrishnan, and L. Kanal (editors). *Parallel Algorithms for Machine Intelligence and Vision*. Springer Verlag, New York, 1990.

[10] Vipin Kumar. Depth-first search. In Stuart C. Shapiro, editor, *Encyclopaedia of Artificial Intelligence: Vol 2*, pages 1004–1005. John Wiley and Sons, Inc., New York, 1987. Revised version appears in the second edition of the encyclopedia to be published in 1992.

[11] Vipin Kumar, Ananth Grama, and V. Nageshwara Rao. Scalable load balancing techniques for parallel computers. Technical report, Tech Report 91-55, Computer Science Department, University of Minnesota, 1991.

[12] Vipin Kumar and Anshul Gupta. Analyzing scalability of parallel algorithms and architectures. Technical report, TR-91-18, Computer Science Department, University of Minnesota, June 1991. A short version of the paper appears in the Proceedings of the 1991 International Conference on Supercomputing, Germany, and as an invited paper in the Proc. of 29th Annual Allerton Conference on Communuication, Control and Computing, Urbana,IL, October 1991.

[13] Vipin Kumar and Anshul Gupta. Analyzing the scalability of parallel algorithms and architectures: A survey. In *Proceedings of the 1991 International Conference on Supercomputing*, June 1991.

[14] Vipin Kumar and Laveen Kanal. Parallel branch-and-bound formulations for and/or tree search. *IEEE Trans. Pattern. Anal. and Machine Intell.*, PAMI–6:768–778, 1984.

[15] Vipin Kumar, K. Ramesh, and V. Nageshwara Rao. Parallel best-first search of state-space graphs: A summary of results. In *Proceedings of the 1988 National Conference on Artificial Intelligence*, pages 122–126, August 1988.

[16] Vipin Kumar and V. N. Rao. Scalable parallel formulations of depth-first search. In Vipin Kumar, P. S. Gopalakrishnan, and Laveen Kanal, editors, *Parallel Algorithms for Machine Intelligence and Vision*. Springer-Verlag, New York, 1990.

[17] Vipin Kumar and V. Nageshwara Rao. Parallel depth-first search, part II: Analysis. *International Journal of Parallel Programming*, 16 (6):501–519, 1987.

[18] Vipin Kumar and Vineet Singh. Scalability of Parallel Algorithms for the All-Pairs Shortest Path Problem: A Summary of Results. In *Proceedings of the International Conference on Parallel Processing*, 1990. Extended version appears in Journal of Parallel and Distributed Processing (special issue on massively parallel computation), Volume 13, 124-138, 1991.

[19] V. Nageshwara Rao and Vipin Kumar. Parallel depth-first search, part I: Implementation. *International Journal of Parallel Programming*, 16 (6):479–499, 1987.

[20] V. Nageshwara Rao and Vipin Kumar. On the efficicency of parallel backtracking. *IEEE Transactions on Parallel and Distributed Systems*, (to appear), 1992. available as a technical report TR 90-55, Computer Science Department, University of Minnesota.

using only 3-dicycle inequalities. Problems whose names start with *diw* are 56 by 56 tables compiled by the Deutches Institut für Wirtschaftforschung for West Germany for the years 1954 to 1972. The last two characters indicate the year; the character before these two indicates whether prices are current prices or 1962 prices. The problems whose names start with *sbm* are compiled by the Statistisches Bundesamt for the years 1974 and 1975 and again are for West Germany. There is also a table available for 1970 but this again can not be solved only using 3-dicycle inequalities. For more details on these problems, see Grötschel et al. [4]. The coefficients $\bar{c}_{ij}$ in these tables are distributed between 0 and approximately 30000, but they are not uniformly distributed: there are a number of entries which are zero and also a number of entries which are small but positive. The zero entries lead to the presence of alternative optima. Interior point methods tend to converge to the interior of the optimal face; therefore, they converge to a solution which corresponds to a *partial* ordering, with every ordering that satisfies the partial ordering being optimal.

The columns in Table 1 have the following meanings. The first column contains the name of the problem. The second column contains the number of 3-dicycle inequalities in the initial relaxation. The third column contains the total number of 3-dicycle inequalities added. The fourth column contains the number of times cutting planes are added. The fifth column contains the number of 3-dicycle inequalities in the final relaxation. The sixth column contains the number of iterations of the primal-dual method which were required. The final column contains the total run time, in seconds. The total number of constraints dropped is the number of constraints in the initial relaxation together with the number of constraints added minus the number of constraints in the final relaxation. The table contains the mean values for each of the three sets of problems.

The runs are sensitive to small changes. For example, using a different routine to calculate the projections increased the number of iterations required to solve sbm1974 from 112 to 128. The two runs are virtually identical through eight stages and 61 iterations, but then one run adds 85 constraints and the other 86 on the ninth stage, with the result that they diverge dramatically.

The number of stages required is similar to the number reported by Grötschel et al. [4] when they added arc-disjoint inequalities; thus, the attempt to identify violated constraints early appears to be successful, without many superfluous constraints being added. The number of iterations per stage is between five and seven, considerably smaller than would be required if each stage was solved to optimality and then the next stage started from scratch.

It is impossible to really compare our run times with those of Grötschel et al. [4], who were using an IBM 370/168 in 1982. For the 44 by 44 tables, their run times are about one minute, for the 56 by 56 they are about four and a half to five minutes, and for the 60 by 60 problems they are almost ten minutes. Thus as the problem size increases, the ratio of their run times to ours increases, which is the standard pattern for simplex results versus interior point results. Now, according to the technical staff at Rensselaer, our machine is about eight to ten times faster per CPU than the IBM 370/168, with the floating point performance ratio probably somewhat higher than this. Therefore, with this simplistic analysis, our runtimes are very competitive with those of Grötschel et al. [4], at least for the larger problems. Note that the simplex method has been improved considerably since 1982 — see for example Bixby et al. [1]. Our runtimes could also be improved by using projection routines designed specifically for interior point methods. For the three problem sets, the time to calculate the

projections was about 20%, 40% and 50% of the total run time, increasing as the size of the problems increased.

Acknowledgement. We are very grateful to Michael Jünger for supplying us with the input-output matrices analyzed in this paper. We are also grateful to Eberhard Kranich for his bibliography on interior point methods.

REFERENCES

[1] R. E. Bixby, J. W. Gregory, I. J. Lustig, R. E. Marsten, and D. F. Shanno. Very large–scale linear programming : A case study in combining interior point and simplex methods. Technical Report J–91–07, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, May 1991. Technical Report SOR 91–08, School of Engineering and Applied Science, Dept. of Civil Engineering and Operations Research, Princeton University, Princeton, NJ 08544, USA, May 1991. Technical Report RRR 34–91, RUTCOR Center of Operations Research, Rutgers University, New Brunswick, NJ 08903, USA, 1991.

[2] B. Borchers. *Improved branch and bound algorithms for integer programming.* PhD thesis, Rensselaer Polytechnic Institute, Mathematical Sciences, Troy, NY, 1992.

[3] J. L. Goffin and J. P. Vial. Cutting planes and column generation techniques with the projective algorithm. *Journal of Optimization Theory and Applications*, 65:409–429, 1990.

[4] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[5] D. den Hertog, C. Roos, and T. Terlaky. A build–up variant of the path–following method for LP. Technical Report 91–47, Faculty of Mathematics and Informatics, TU Delft, NL–2628 BL Delft, The Netherlands, June 1991.

[6] IBM. *Engineering and Scientific Subroutine Library Guide and Reference.* Kingston, NY, 1988.

[7] M. Jünger. *Polyhedral Combinatorics and the Acyclic Subdigraph Problem.* Heldermann, Berlin, 1985.

[8] J. A. Kaliski and Y. Ye. A decomposition variant of the potential reduction algorithm for linear programming. Working Paper, Dept. of Management Science, College of Business Administration, University of Iowa, Iowa City, IA 52240, USA, 1991.

[9] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Computational experience with a primal–dual interior point method for linear programming. *Linear Algebra and Its Applications*, 152:191–222, 1991.

[10] J. E. Mitchell and M. J. Todd. Solving combinatorial optimization problems using Karmarkar's algorithm. *Mathematical Programming*, to appear.

[11] G. Reinelt. *The Linear Ordering Problem: Algorithms and Applications.* Heldermann, Berlin, 1985.

[12] M. G. C. Resende and G. Veiga. Computational investigation of an interior point linear programming algorithm for minimum cost network flow. Technical Report 1.0, Mathematical Sciences Research Center, AT & T Bell Laboratories, Murray Hill, NJ 07974, USA, August 1991.

3. Parallel Best First Search. The A* algorithm is a well known search algorithm that can use problem-specific heuristic information to prune search space. As discussed in [10], A* is essentially a "best-first" branch-and-bound (B&B) algorithm. A number of researchers have investigated parallel formulations of A*/B&B algorithms [14,15]. An important component of A*/B&B algorithms is the priority queue which is used to maintain the "frontier" (*i.e.*, unexpanded) nodes of the search graph in a heuristic order. In the sequential A*/B&B algorithm, in each cycle a most promising node from the priority queue is removed and expanded, and the newly generated nodes are added to the priority queue.

In most parallel formulations of A*, different processors concurrently expand different frontier nodes. Conceptually, these formulations can be viewed to differ in the data structures used to implement the priority queue of the A* algorithm. Some formulations are suited only for shared-memory architectures, whereas others are suited for distributed-memory architectures as well. The effectiveness of different parallel formulations is also strongly dependent upon the characteristics of the problem being solved. We have investigated a number of parallel formulations of A*[15]. Some of these formulations are new, and the others are very similar to the ones developed by other researchers. We have tested the performance of these formulations on the 15-puzzle, the traveling salesman problem(TSP), and the vertex cover problem (VCP) on the BBN Butterfly^{TM} multiprocessor. The results for the 15-puzzle and VCP are very similar, but very different from results obtained for the TSP. The reason is that the TSP and VCP generate search spaces that are qualitatively different from each other, even though both problems are NP-hard problems. We have also performed a preliminary analysis of the relationship between the characteristics of the search spaces and their suitability to various parallel formulations[15].

4. Speedup Anomalies in Parallel Search. In parallel DFS, the speedup can differ greatly from one execution to another, as the actual parts of the search space examined by different processors are determined dynamically, and can be different for different executions. Hence, for some execution sequences the parallel version may find a solution by visiting fewer nodes than the sequential version thereby giving superlinear speedup, whereas for others it may find a solution only after visiting more nodes resulting in sublinear speedup. It may appear that on the average the speedup would be either linear or sublinear. This phenomenon of speedup greater than P on P processors in isolated executions of parallel DFS has been reported by many researchers. for a variety of problems and is referred to by the term speedup anomaly.

In [20], we present analytical models and experimental results on the average case behavior of parallel backtracking. We consider two types of backtrack search algorithms: (i) simple backtracking (which does not use any heuristic information); (ii) heuristic backtracking (which uses heuristics to order and prune search). We present analytical models to compare the average number of nodes visited in sequential and parallel search for each case. For simple backtracking, we show that the average speedup obtained is (i) linear when distribution of solutions is uniform and (ii) superlinear when distribution of solutions is non-uniform. For heuristic backtracking, the average speedup obtained is at least linear (i.e., either linear or superlinear), and the speedup obtained on a subset of instances (called difficult instances) is superlinear. We also present experimental results over many synthetic and practical problems on various parallel machines, that validate our theoretical analysis.

TABLE 2.1
Scalability results of various parallel DFS formulations for different architectures.

Scheme→ Arch↓	ARR	NN	GRR	GRR-M	Random	Lower Bound
SM	$O(P^2 \log P)$	$O(P^2 \log P)$	$O(P^2 \log P)$	$O(P \log P)$	$O(P \log^2 P)$	$O(P)$
Cube	$O(P^2 \log^2 P)$	$\Theta(P^{\log_2 \frac{1+\frac{1}{\alpha}}{2}})$	$O(P^2 \log P)$	$O(P \log^2 P)$	$O(P \log^3 P)$	$O(P \log P)$
Ring	$O(P^3 \log P)$	$\Theta(K^P)$	$O(P^2 \log P)$		$O(P^2 \log^2 P)$	$O(P^2)$
Mesh	$O(P^{2.5} \log P)$	$\Theta(K^{\sqrt{P}})$	$O(P^2 \log P)$	$O(P^{1.5} \log P)$	$O(P^{1.5} \log^2 P)$	$O(P^{1.5})$
Network of workstations	$O(P^3 \log P)$	$O(P^3 \log P)$	$O(P^2 \log P)$		$O(P^2 \log^2 P)$	$O(P^2)$

head incurred by different processors in finding work and in contention over shared resources. By modeling these, we were able to determine the isoefficiency functions (and thus scalability) of a variety of work-distribution schemes[7,11,17] for the ring, cube, mesh, network of workstations, and shared memory architectures. Some of the existing parallel formulations were found to have poor scalability. This motivated the design of substantially improved schemes for various architectures. We established lower bounds on the scalability of any possible parallel formulation for various architectures. For each of these architectures, we determined near optimal load balancing schemes. The performance characteristics of various schemes have been proved and experimentally verified in [7,11,17,19]. Schemes with better isoefficiency functions performed better than those with poorer isoefficiency functions for a wide range of problem sizes and processors. Table 2.1 presents isoefficiency functions of various parallel formulations on various architectures. For a more detailed discussion see [7,11].

From our scalability analysis of a number of architecture-algorithm combinations, we have been able to gain valuable insights into the relative performance of parallel formulations for a given architecture. For instance, in [3] an implementation of parallel depth first branch and bound for VLSI floorplan optimization is presented, and speedups obtained on a network of 16 workstations. The essential part of this branch-and-bound algorithm is the GRR load balancing technique. Our scalability analysis can be used to investigate the viability of using a much larger number of workstations for solving this problem. Note from Table 2.1 that GRR has an overall isoefficiency of $O(P^2 \log P)$ for this platform. Hence, if we had 1024 workstations on the network, we can obtain the same efficiency on a problem instance which is 10240 ($= \frac{1024^2 \log 1024}{16^2 \log 16}$) times bigger compared to a problem instance being run on 16 processors. This result is of significance, since it indicates that it is indeed possible to obtain good efficiencies with large number of workstations. Scalability analysis also sheds light on the degree of scalability of such a system with respect to other parallel architectures such as hypercube and mesh multicomputers. For instance, the best applicable technique implemented on a hypercube has an isoefficiency function of $O(P \log^2 P)$. With this isoefficiency, we would be able to get identical efficiencies as those obtained on 16 processors by increasing the problem size 400 fold (which is $\frac{1024 \log^2 1024}{16 \log^2 16}$). We can thus see that it is possible to obtain good efficiencies even with smaller problems on the hypercube. We can thus conclude from isoefficiency functions that the hypercube offers a much more scalable platform compared to the network of workstations for this problem.

TEST CASES FOR THE MAXIMUM CLIQUE PROBLEM

PANOS M. PARDALOS* AND GEORGE VAIRAKTARAKIS*

Abstract. In this note we present a number of test cases for the maximum clique problem. The instances presented are taken from diverse areas of applications including coding theory, fault diagnosis, computational geometry, as well as from specially generated graphs. These test problems can be used to prove correctness of algorithms and in computational comparisons.

1. Introduction. Throughout this paper, we denote by $G = (V, E)$ an undirected graph with vertex set $V = \{1, 2, \ldots, n\}$ and edge set $E \subseteq V \times V$. A graph is *complete* if $\forall\ i, j \in V,\ (i, j) \in E$. A *clique* C of G is a complete subgraph of G. The maximum clique problem asks for a clique of maximum cardinality. Related problems are the maximum independent set and the minimum vertex cover problem.

Over the last years a great number of researchers have spent much effort on attacking this extremely intractable problem. By now, many theoretical results and several exact as well as heuristic algorithms are known for this problem. An exhaustive survey is presented in [11] where an up-to-date bibliography is provided. Some of the most computationally efficient exact algorithms for the maximum clique problem are given in [1], [2], [5], [10]. However, none of the existing exact algorithms can handle dence instances having more than a thousand vertices.

Up to this date, the test generators that have been proposed for the maximum clique problem are random in nature (see [9]). In this paper we present instances which include problems from coding Theory, Keller's conjecture and fault diagnosis.

2. Problems Arising from Coding Theory. One of the fundamental questions in Coding Theory [8] is to find the number of binary vectors of size n with Hamming distance d. We denote this number by $A(n, d)$. The Hamming distance $dist(u, v)$ between the binary vectors $u = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2, \ldots, v_n)$ is the number of indices i such that $1 \leq i \leq n$ and $u_i \neq v_i$. It is known that a set of binary vectors any two of which have distance greater or equal to d, can correct $\lfloor \frac{d-1}{2} \rfloor$ errors. Without loss of generality we may always assume that d is even.

2.1. Hamming Graphs. The above problem can be restated as a maximum clique problem as follows. We define the Hamming graph of size n and distance d, denoted by $H(n, d)$, as the graph with vertex set the binary vectors of size n, and two vertices are connected by an edge if their Hamming distance is at least d. Then, $A(n, d)$ is the size of a maximum clique in $H(n, d)$. It is easy to see that $H(n, d)$ has 2^n vertices, $2^{n-1} \sum_{i=d}^{n} \binom{n}{i}$ edges, and the degree of each vertex is $\sum_{i=d}^{n} \binom{n}{i}$. The interested reader can find all results known to date for $A(n, d)$ with $n \leq 28$ in [14] and [4].

2.2. Johnson Graphs. Another fundamental question in Coding Theory is to find constant weight codes, that is, to find the maximum number of binary vectors of size n, any two of which have Hamming distance d and each vector contains precisely w ones. This number is denoted by $A(n, w, d)$.

The corresponding graph representation is as follows. Form the Johnson Graph $J(n, w, d)$ which consists of $\binom{n}{w}$ vertices labeled by binary vectors of length n. Two vertices are adjacent in $J(n, w, d)$ if their labels have Hamming distance at least d.

* Department of Industrial and Systems Engineering, University of Florida, Gainesville FL, 32611, USA

Then, $A(n, w, d)$ is the size of a maximum clique of $J(n, w, d)$. It is known (see [14]) that a code of size n and weight w can correct $w - \frac{d}{2}$ errors. All known lower bounds and exact values for constant weighted codes with distance between 2 and 18 are given in [4]. The interested reader can refer to this paper for all details.

3. Problems Arising from Keller's Conjecture. A family of hypercubes with disjoint interiors whose union is R^n is a tiling. Lattice tiling is a tiling for which the centers of the hypercubes form a lattice.

In the beginning of the century, Minkowski conjectured that in a lattice tiling of R^n by translates of a unit hypercube, there exist two cubes that share a $(n-1)$-dimensional face. About fifty years later, Hajös proved Minkowski's conjecture. At 1930, Keller suggested that Minkowski's conjecture holds even in the absence of the lattice assumption. Ten years later Perron proved correctness of Keller's conjecture for $n \leq 6$. Since then, many efforts have been devoted to prove or disprove this conjecture. Recently, a combinatorial equivalent of the conjecture was given by Corrádi and Szabó [6] which states that: *There is a counterexample to Keller's conjecture if and only if there exists $n \in N$ for which the graph Γ_n has a clique of size 2^n.* The class of Keller's graphs Γ_n is described next.

3.1. Keller Graphs. Consider the set $V_n = \{(d_1, \ldots, d_n) : d_i \in \{0, 1, 2, 3\}\ for\ 1 \leq i \leq n\}$ and define the graph Γ_n with vertex set V_n and edge set as follows: Given a graph G, let $(v \perp_G u)$ denote that vertices v, u are adjacent in G. Two nodes $v = (d_1, \ldots, d_n)$ *and* $u = d'_1, \ldots, d'_n$ in V_n are adjacent iff $d_i - d'_i \equiv 2 \bmod 4$ for some $1 \leq i \leq n$ and $d_j \neq d'_j$ for some $1 \leq j \leq n$ and $i \neq j$.

Clearly, Γ_n has 4^n vertices. It is not hard to show that every vertex $v \in V_n$ has degree $d(v) = 4^n - 3^n - n$. Also, one can verify that if $E(\Gamma_n)$ is the edge set of Γ_n, then Γ_n has density $\frac{|E(\Gamma_n)|}{|E(K_{4^n})|} = \frac{4^n - 3^n - n}{4^n - 1}$ where K_m denotes the complete graph on m vertices. A quick calculation shows that Γ_7 has density exceeding 0.866 and Γ_{15} has density over 0.9866 ! Recently it has been proved that for $n \geq 10$ the clique size of Γ_n is $cl(\Gamma_n) = 2^n$. This means that Keller's conjecture fails for $n \geq 10$ but it remains open for $n \in \{7, 8, 9\}$. The following table (see [6]) gives the maximum clique size of the graphs Γ_n for $n \leq 6$.

n	2	3	4	5	6
2^n	4	8	16	32	64
$cl(\Gamma_n)$	2	5	12	28	57

TABLE 3.1
Values of $cl(\Gamma_n)$ for $n \leq 6$

4. Problems Arising From Fault Diagnosis. Over the last years large multiprocessor systems have gained an essential role in computing. For this matter, reliability of these systems is very important and diagnosis of faulty machines becomes a problem. The classical approach on fault diagnosis originated by Preparata, Metze and Chien twenty years ago by a model known as PMC (see [3]). In this model it is assumed that each unit can test some other units and that fault-free units always detect correct results while faulty units can output any result. Also, it is assumed that the number of faulty units does not exceed a certain bound t (t cannot exceed the number of neighbors of any unit) and gathered results are transmitted and processed by a monitoring unit from which we ascertain the state of the system.

MIMD multiprocessor. In these parallel formulations, the search space is dynamically distributed among a number of processors. These formulations differ in the methods used for distributing work among processors. To study the effectiveness of these formulations, we have applied these to a number of problems. Parallel formulations of IDA* for solving the 15 puzzle problem yielded near linear speedup on Sequent Balance up to 30 processors and on the Intel Hypercube^{TM} and BBN Butterfly^{TM} up to 128 processors [16,17,19]. Parallel formulation of PODEM, which is the best known sequential algorithm for solving the test pattern generation problem, provided linear speedup on a 128 processor Symult^{TM} [5]. Parallel depth-first branch-and-bound for floorplan optimization for VLSI circuits yielded linear speedups on a 1024 processor Ncube^{TM}, a 128 processor Symult^{TM} and a network of 16 SUN workstations [3]. Linear speedups were also obtained for parallel DFS for the tautology verification problem for up to 1024 processors on the Ncube/10^{TM} and the Ncube/2^{TM}[4,7,11].[1] In [8], we have presented new methods for load balancing of unstructured tree computations on large-scale SIMD machines such as CM-2 ^{TM}[2]. The analysis and experiments show that our new load balancing methods provide good speedups for parallel DFS on SIMD architectures. In particular, their scalability is no worse than that of the best load balancing schemes on MIMD architectures.

Scalability Analysis. From experimental results for a particular architecture and a range of processors alone, it is difficult to ascertain the relative merits of different parallel algorithm-architecture combinations. This is because the performance of different schemes may be altered in different ways by changes in hardware characteristics (such as interconnection network, CPU speed, speed of communication channels etc.), number of processors, and the size of the problem instance being solved [12]. Hence any conclusions drawn from experimental results on a specific parallel computer and problem instance are rendered invalid by changes in any one of the above parameters. Scalability analysis of a parallel algorithm and architecture combination has been shown to be useful in extrapolating these conclusions [13,17].

We have developed a scalability metric, called *isoefficiency* , which relates the problem size to the number of processors necessary for an increase in speedup in proportion to the number of processors used [17]. In general, for a fixed problem size W, increasing the number of processors P causes a decrease in efficiency because parallel processing overhead will increase while the sum of time spent by all processors in meaningful computation will remain the same. Parallel systems that can maintain a fixed efficiency level while increasing both W and P are defined as *scalable* [13]. If W needs to grow as $f(P)$ to maintain an efficiency E, then $f(P)$ is defined to be the **isoefficiency function** for efficiency E and the plot of $f(P)$ with respect to P is defined to be the **isoefficiency curve** for efficiency E. An important feature of isoefficiency analysis is that it succinctly captures the effects of characteristics of the parallel algorithm as well as the parallel architecture on which it is implemented, in a single expression. The isoefficiency metric has been found to be very useful in characterizing scalability of a number of algorithms [18].

Scalability Analysis of Parallel Depth-First Search. The primary reason for loss of efficiency in our parallel formulations of DFS is the communication over-

[1] Our work on tautology verification, test pattern generation and floorplan optimization received honorable mention for the Gordon Bell Award for outstanding research in practical parallel computing[6]. This is significant considering that this award has historically been given to researchers working on numerical problems.

[2] CM-2 is a registered trademark of the Thinking Machines Corporation.

PARALLEL ALGORITHMS FOR DISCRETE OPTIMIZATION PROBLEMS*

V. KUMAR† AND GRAMA Y. ANANTH†

1. Introduction. Discrete optimization problems (DOPs) arise in various applications such as planning, scheduling, computer aided design, robotics, game playing and constraint directed reasoning. Often, a DOP is formulated in terms of finding a (least cost) solution path in a graph from an initial node to a goal node and solved by graph/tree search methods such as branch-and-bound and dynamic programming.

Parallel computers containing thousands of processing elements are now commercially available. The cost of these machines is similar to that of large mainframes, but they offer significantly more raw computing power. Due to advances in VLSI technology and economy of scale, the cost of these machines is expected to go down drastically over the next decade. It may be possible to construct computers comprising of thousands to millions of processing elements at costs ranging from those of high-end workstations to large mainframes. This technology has created substantial interest in exploring the use of parallel processing for search based applications [1,2,9,14].

Given that DOP is an NP-hard problem, one may argue that there is no point in applying parallel processing to these problems, as the worst-case run time can never be reduced to a polynomial unless we have an exponential number of processors. However, the average time complexity of heuristic search algorithms for many problems is polynomial. Also, there are some heuristic search algorithms which find suboptimal solutions in polynomial time (e.g., for certain problems, approximate branch-and-bound algorithms are known to run in polynomial time). In these cases, parallel processing can significantly increase the size of solvable problems. Some applications using search algorithms (e.g. robot motion planning, task scheduling) require real time solutions. For these applications, parallel processing is perhaps the only way to obtain acceptable performance. For some problems, optimal solutions are highly desirable, and can be obtained for moderate sized instances in a reasonable amount of time using parallel search techniques (e.g. VLSI floor-plan optimization [3]).

In this article, we provide an overview of our research on parallel algorithms for solving discrete optimization problems.

2. Parallel Depth First Search. Depth-first search (DFS), also referred to as Backtracking, is a general technique for solving a variety of discrete optimization problems [10,5,3]. Since many of the problems solved by DFS are computationally intensive, there has been a great interest in developing parallel versions of DFS. For most applications, state space trees searched by DFS tend to be highly irregular, and any static allocation of subtrees to processors is bound to result in significant load imbalance among processors.

In [7,11,17,19], we presented and analyzed a number of parallel formulations of DFS which retain the storage efficiency of DFS, and can be implemented on any

* This work was supported by IST/SDIO through the Army Research Office grant # 28408-MA-SDI to the University of Minnesota and by the Army High Performance Computing Research Center at the University of Minnesota.

† Department of Computer Science, University of Minnesota, Minneapolis, MN 55455 – (612) 625-4002 – *kumar@cs.umn.edu* and *ananth@cs.umn.edu*

The assumptions of the PMC model are prohibitive and unrealistic. In particular, economic considerations force large systems to have low connectivity which means that it is very likely the bound t to exceed the number of neighbors of some unit.

These observations led researchers to more realistic models one of which was introduced by Berman and Pelc [3]. For any number of units, n, they designed a system G_n with $\mathcal{O}(n \log n)$ links. It is assumed that every unit tests the units it is linked to, and that a fault-free unit fails to detect a fault unit with probability q, while fault-free units never detect faults in each other. For the proposed design G_n, it is shown that the diagnosis is correct with probability $1 - n^{-1}$.

4.1. C-Fat Rings. For fixed n and given parameter c, the graph $G_n = (V, E)$ is called c-fat ring and it is defined as follows: The vertex set V is the set of n units. Let $k = \lfloor n/c \log n \rfloor$. We partition the vertex set into k parts $W_0, \ldots, W_{k-1}$ so that

$$c \log n \leq |W_i| \leq 1 + \lceil c \log n \rceil \quad i = 0, 1, \ldots, k-1.$$

We can assume that the number of leading parts have precisely $\lceil c \log n \rceil + 1$ vertices, while the rest have $\lceil c \log n \rceil$ vertices. The adges of the graph are defined as follows:

$$For \;\; u \in W_i \;\; and \;\; v \in W_j \;\; (u,v) \in E \;\; iff \;\; u \neq v \;\; and \;\; |i-j| \in \{0, 1, k-1\},$$

which means that the vertex is adjacent to vertices in the same or ajacent partitions. The key step to the algorithm proposed in [3] is to find a maximum clique of the c-fat ring.

5. Test Generator Graphs. In this section we present a set of instances proposed in [13] for the vertex cover problem. The vertex cover problem is to find a minimum cardinality subset $V^\star$ of the vertex set V of $G = (V, E)$ such that every edge in E is incident to at least one vertex in $V^\star$. This problem is equivalent to solving the maximum clique problem on the complement of G, denoted by $\bar{G}$. The complement of $G = (V, E)$ is the graph $\bar{G} = (V, \bar{E})$ where $\bar{E} = \{(i,j) | i, j \in V,\ i \neq j \;\; and \;\; (i,j) \notin E\}$. Thus, if G is a graph of size n and known minimum vertex cover size c, the complement $\bar{G}$ has size n and maximum clique size $cl(\bar{G}) = n - c$. To generate $G = (V, E)$ with $|V| = n$, $|E| = m$ and vertex cover c, according to [13], we do as follows.

Compute $k = n - c$ and partition n into k parts, $n = n_1 + n_2 + \ldots + n_k$ such that $m' = \sum_{i=1}^{k} \binom{n_i}{2} \leq m$. Then form k cliques of sizes $n_1, n_2, \ldots, n_k$. For each $1 \leq i \leq k$, choose any $n_i - 1$ vertices from the $i - th$ clique to put in the vertex cover. Add $m - m'$ additional edges to the graph in such a way that each added edge is incident to at least one of the selected cover vertices. Under the conditions that

$$0 \leq c \leq n - 1$$

and

$$r\binom{b+1}{2} + (k-r)\binom{b}{2} \leq m \leq \binom{c}{2} + kc$$

where $k = n - c$ and $n = kb + r$ (i.e., b and r are the quotient and the remainder when n is divided by k) it is shown that the resulting graph has minimum vertex cover of size c and therefore its complement has maximum clique of size $n - c$. The advantage of this family of instances is that the size of the vertex set, the size of the edge set as well as the maximum clique size are predetermined.

6. Remarks. Test problems are of practical importance in evaluating algorithms or heuristics and performing computational comparisons. In this note we presented a number of test instances that have interesting applications. Computational results, as well as the codes of the test generators that we describe here, are presented in detail in [7].

REFERENCES

[1] Balas, E. and J. Xue, Minimum Weighted Coloring of Triangulated Graphs, with Application to Maximum Weight Vertex Packing and Clique Finding in Arbitrary Graphs, *SIAM J. Computing* 20(2):209-221, 1991.

[2] Balas, E. and C.S. Yu, Finding a Maximum Clique in an Arbitrary Graph, *SIAM J. Computing* 14(4):1054-1068, 1986.

[3] Berman P. and A. Pelc, Distributed Fault Diagnosis for Multiprocessor Systems, *Proc. of the 20th Annual Intern. Symp. on Fault-Tolerant Computing* (Newcastle, UK) 340-346, 1990.

[4] Brouwer, A. E., J. B. Shearer, N. J. A. Sloane and W. D. Smith. A New Table of Constant Weight Codes. *IEEE Transactions on Information Theory* 36(6):1334-1380, 1990.

[5] Carraghan R. and P.M. Pardalos, An Exact Algorithm for the Maximum Clique Problem, *Operations Research Letters* 9:375-382, 1990.

[6] Corrádi, K. and S. Szabó. A combinatorial approach for Keller's conjecture, *Periodica Math. Hung.* 21(2):95-100, 1990.

[7] Hasselberg J., P.M. Pardalos and G. Vairaktarakis, Test Case Generators and Computational Results for the Maximum Clique Problem, *Journal of Global Optimization*, 1992.

[8] MacWilliams F. J. and N. J. A. Sloane. The Theory of Error-Correcting Codes. *North Holland, Amsterdam* 1979.

[9] Pardalos P.M., Construction of test problems in quadratic bivalent programming, *ACM Transactions on Mathematical Software* 17(1): 74-87, 1991.

[10] Pardalos P.M. and G.P. Rodgers, A Branch and Bound Algorithm for the Maximum Clique Problem, *Computers and Operations Research* 19(5):363-375, 1992.

[11] Pardalos P.M. and J. Xue, The Maximum Clique Problem, *Journal of Global Optimization*, 1992.

[12] Perron O., Über lückenlose Ausfullung des n-dimensionalen Raumes durch kongruente Würfel, *Math. Z.* 46:1-26,161-180, 1940.

[13] Sanchis L., Test Case Construction for the Vertex Cover Problem (extended abstract), DIMACS Workshop on Computational Support for Discrete Mathematics, March 1992.

[14] Sloane N. J. A., Unsolved Problems in Graph Theory Arising from the Study of Codes. *Graph Theory Notes of New York XVIII* 11-20, 1989.

in practice, lower bounds do not turn out as sharp as they could potentially be (for instance, at least 2 out of the 3 test problems for which we report a positive duality gap, could be solved to optimality, at node 0 of a search tree, with a branch and cut algorithm that uses the same inequalities as we do). The main attraction here is that our lower bounds are still very sharp and the time to compute them is, relatively speaking, low. One factor that helps explain this is the low computational complexity associated with the detection of violated inequalities. This may prove even more attractive for applications where the separation problem, for a given type of valid inequality, in an equivalent branch and cut algorithm, could only be tackled, due to their computational complexity, by approximation algorithms.

REFERENCES

[1] A. Balakrishnan and N.R. Patel. Problem reduction methods and a tree generation algorithm for the Steiner problem in graphs. *Networks*, 17:86–85, 1987.

[2] J.E Beasley. An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989.

[3] S. Chopra, E.R. Gorres, and M.R. Rao. Solving the Steiner tree problem on a graph using branch and cut. Technical report, Department of Managerial Economics, J.L. Kellogg Graduate School of Management, Northwestern University, Evanston, IL 60208, 1991.

[4] S. Chopra and M.R. Rao. The Steiner tree problem I: formulations, compositions and extension of facets. Technical report, Department of Statistics and Operations Research, Leonard N. Stern School of Business, New York University, New York, NY 10006, 1988.

[5] J.J. Dongarra. Performance of various computers using standard linear equations software. Technical report, Computer Science Department, University of Tennessee, Knoxville, TN 37996-1301, 1991.

[6] M.L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. Technical report, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, 1990.

[7] M.X. Goemans. The Steiner tree polytope and related polyhedra. Technical report, Department of Mathematics, MIT, Cambridge, MA 02139, 1991.

[8] A. Lucena. Tight bounds for the Steiner problem in graphs, July 15-17 1991. Talk given at the TIMS XXX - SOBRAPO XXIII Joint International Meeting, Rio de Janeiro.

[9] A. Lucena and J.E. Beasley. A branch and cut algorithm for the Steiner problem in graphs. Technical report, The Management School, Imperial College, London SW7 2AZ, 1991.

[10] F. Margot, A. Prodon, and Th. M. Liebling. Tree polyhedron on 2-tree. Technical report, Ecole Polytechnique Federale de Lausanne, Lausanne, 1991.

[11] M.L.Fisher. The Lagrangean relaxation method for solving integer programming problems. *Management Science*, 27:1–18, 1981.

[12] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[13] V.J. Rayward-Smith and A. Clare. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *Networks*, 16:283–294, 1986.

[14] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 6:573–577, 1980.

TABLE 4.2
Problems D

Problem	$\|E\|$	$\|T\|$	$\max C_3$	$\max C_5$	Cycles	Bounds		Optimal	CPU time
						Lower	Upper	Value	(secs)
1	1250	5	39	73	2	106	106	106	101
2		10	60	126	1	220	220	220	72
3		167	123	193	1	1565	1565	1565	67
4		250	83	102	1	1935	1935	1935	42
5		500	17	25	1	3250	3250	3250	31
6	2000	5	86	97	7	63.14	67	67	1546
7		10	61	93	1	103	103	103	351
8		167	327	739	2	1072	1072	1072	1107
9		250	358	617	2	1448	1448	1448	800
10		500	181	220	1	2110	2110	2110	294
11	5000	5	51	58	2	29	29	29	1026
12		10	69	113	2	42	42	42	844
13		167	352	667	1	500	500	500	1215
14		250	350	583	2	667	667	667	1318
15		500	300	311	1	1116	1116	1116	763
16	25000	5	44	31	1	13	13	13	630
17		10	76	109	1	23	23	23	1039
18		167	344	582	1	223	223	223	1679
19		250	363	489	2	310	310	310	1477
20		500	44	34	1	537	537	537	1238

TABLE 4.3
Problems E

Problem	$\|E\|$	$\|T\|$	$\max C_3$	$\max C_5$	Cycles	Bounds		Optimal	CPU time
						Lower	Upper	Value	(secs)
1	3125	5	43	58	1	111	111	111	232
2		10	60	139	2	214	214	214	479
3		417	339	528	1	4013	4013	4013	549
4		625	213	242	1	5101	5101	5101	379
5		1250	40	40	1	8128	8128	8128	426
6	5000	5	61	78	2	73	73	73	2337
7		10	119	259	5	145	145	145	5392
8		417	779	1433	4	2640	2640	2640	13498
9		625	814	1372	3	3604	3604	3604	8854
10		1250	558	542	2	5600	5601	5600	3933
11	12500	5	91	83	2	34	34	34	6449
12		10	121	214	4	62.59	67	67	26333
13		417	889	1691	3	1280	1281	1280	39445
14		625	909	1401	6	1732	1732	1732	23341
15		1250	718	794	2	2784	2784	2784	8916
16	65000	5	67	29	1	15	15	15	3937
17		10	108	140	2	25	25	25	9682
18		417	940	1621	2	562.1	568	?	46762
19		625	675	1028	4	758	759	758	25690
20		1250	383	369	1	1342	1347	1342	19930

AN IMPROVEMENT ON KARMARKAR'S ALGORITHM FOR INTEGER PROGRAMMING*

C.-J. SHI†, A. VANNELLI‡, AND J. VLACH‡

Abstract. In this note, we propose an improvement of Karmarkar's interior point method for integer programming and present our computational experience with its application to the Boolean Satisfiability problem. We first observe that the step size used in Karmarkar's algorithm can be larger than 1 and that it affects both the number of iterations and the number of linear system solutions significantly. We apply the line search technique to choose a local optimal step size. Experiments with the Boolean Satisfiability problem indicate that the use of the these more ideal step sizes reduce the number of iterations by 5-15 times, and the number of linear system solutions by 2-6 times, with only about one percent extra time for the line search.

1. Introduction. Karmarkar [3] shows that the linear programming problem can be solved by an interior point approach in polynomial time. Recent research by Karmarkar and his colleagues has been made to extend the interior point approach to solve difficult NP-complete problems [2,4,5]. The basic idea is to formulate a 0-1 integer programming problem as a quadratic optimization problem over polytopes, which is then solved by the interior point method. In order to use the interior point method, a potential function is constructed and a sequence of interior points is generated to minimize the potential function. At each point in the sequence, the descent direction is determined by optimizing a local quadratic approximation to the potential function over a search region, which is similar to the search region used in linear programming. The *step size* in the direction of steepest descent is chosen in the interval $(0, 1]$.

In this note, we propose a practically significant improvement to the above approach for solving 0-1 integer programming problems. We observe that the step size can be an arbitrary real number an thus apply a line search technique to choose the optimal step size. We compare our results with this new line search approach with the approach described in Kamath *et al.* [2] on the Satisfiability problem. Not only were the results from their original paper reproduced, our computational experience indicates that with less than one percent extra time for line search, we reduce the iteration number by orders of magnitude.

2. Interior Point Method for Integer Programming. For simplicity throughout this short note we focus on the Boolean Satisfiability problem. The interested reader is referred to Karmarkar *et al.* [5] and Kamath *et al.* [2] for details on the algorithm and the application to Satisfiability.

Consider the following binary integer programming problem: Find a feasible solution $\mathbf{x}$ where $x_i \in \{-1, +1\}, \forall\, i = 1, 2, \ldots, n$ and

$$\hat{\mathbf{A}}\mathbf{x} \leq \hat{\mathbf{b}}, \tag{2.1}$$

where $\hat{\mathbf{A}} \in R^{\hat{m} \times n}$ and $\hat{\mathbf{b}} \in R^{\hat{m}}$.

* This research was partially supported by an operating grant (No. OGP 0044456), a Micronet Research Fund grant from the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario. This paper was initially presented at the *Sixth SIAM Conference on Discrete Mathematics*, University of British Columbia, Vancouver, CANADA, June 8-11, 1992.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, CANADA N2L 3G1

‡ Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, CANADA N2L 3G1

The basic interior point method used by Kamath *et al.* [2] and Karmarkar *et al.* [5] to solve problem (2.1) involves several steps. First, problem (2.1) is transformed into the equivalent continuous quadratic programming problem

$$\text{Maximize } \mathbf{x}^T\mathbf{x}$$

$$\text{s.t. } \hat{\mathbf{A}}\mathbf{x} \leq \hat{\mathbf{b}} \tag{2.2}$$

$$-\mathbf{e} \leq \mathbf{x} \leq \mathbf{e}$$

where $\hat{\mathbf{A}} \in R^{\hat{m}\times n}$ and $\hat{\mathbf{b}} \in R^{\hat{m}}$, $\mathbf{x} \in R^n$, $\mathbf{e} \in R^n$ and $\mathbf{e}^T = (1, 1, \ldots, 1)$.

The feasible region of (2.2) is the polytope $\mathcal{P}$ where

$$\mathcal{P} = \{\mathbf{x} \in R^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b} \text{ and } -\mathbf{e} \leq \mathbf{x} \leq \mathbf{e}\}$$

where $\mathbf{b}^T = (\hat{\mathbf{b}}^T, 1, 1, \cdots, 1)$, $\mathbf{I}$ is an $n \times n$ identity matrix, $\mathbf{A} \in R^{m\times n}$, $\mathbf{A}^T = [\hat{\mathbf{A}}^T \mid \mathbf{I} \mid -\mathbf{I}]$,
$\mathbf{b} \in R^m$ and $m = 2n + \hat{m}$. Problem (2.2) is NP-complete [8].

The general strategy to solve (2.2) is to choose an interior point $\mathbf{x}^0 \in \mathcal{P}$ and generate a sequence of interior points $\mathbf{x}^k \in \mathcal{P}$ such that the potential function $\psi(\mathbf{x})$ given by

$$\psi(\mathbf{x}) = \log\sqrt{n - \mathbf{x}^T\mathbf{x}} - \eta\sum_{j=1}^{m} \log\, d_j(\mathbf{x}) \tag{2.3}$$

is minimized. In equation (2.3), $d_j(\mathbf{x}) = b_j - \sum_{i=1}^{n} a_{ji}x_i$ and η is a parameter which is set to $\frac{1}{n}$ in [5,2]. If $\Delta\mathbf{x}$ is descent direction of $\psi(\mathbf{x})$ around $\mathbf{x}^k$, then a possible sequence of interior points can be generated by updating

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha\Delta\mathbf{x}, \tag{2.4}$$

where α is a step size.

Karmarkar *et al.* [5] show that the steepest descent direction $\Delta\mathbf{x}$ can be determined by solving the following quadratic programming problem:

$$\text{Minimize } \frac{1}{2}(\Delta\mathbf{x})^T\mathbf{H}\Delta\mathbf{x} + \mathbf{h}^T\Delta\mathbf{x}$$

$$s.t.\ (\Delta\mathbf{x})^T\mathbf{A}\mathbf{D}^2\mathbf{A}^T(\Delta\mathbf{x}) \leq r^2 \leq 1 \tag{2.5}$$

where the Hessian $\mathbf{H}$ is

$$\mathbf{H} = -\frac{2}{f_0}\mathbf{I} - \frac{4}{f_0^2}\mathbf{x}^k(\mathbf{x}^k)^T + \frac{1}{n}\mathbf{A}\mathbf{D}^2\mathbf{A}^T,$$

and the gradient is

$$\mathbf{h} = -\frac{1}{f_0}\mathbf{x}^k + \frac{1}{n}\mathbf{A}\mathbf{D}\mathbf{e}.$$

Problem (2.5) is solved in *polynomial time* [9].

TABLE 4.1
Problems C

Problem	$\|E\|$	$\|T\|$	max C_3	max C_5	Cycles	Bounds		Optimal	CPU time
						Lower	Upper	Value	(secs)
1	625	5	21	54	1	85	85	85	16
2		10	37	117	2	144	144	144	37
3		83	55	114	1	754	754	754	20
4		125	59	113	1	1079	1079	1079	20
5		250	12	18	1	1579	1579	1579	5
6	1000	5	40	54	2	55	55	55	147
7		10	54	94	1	102	102	102	83
8		83	161	356	1	509	509	509	167
9		125	179	498	2	707	707	707	306
10		250	64	78	1	1093	1093	1093	35
11	2500	5	39	61	2	32	32	32	201
12		10	65	100	1	46	46	46	216
13		83	163	432	3	258	258	258	431
14		125	154	298	1	323	323	323	191
15		250	153	179	1	556	556	556	130
16	12500	5	28	48	1	11	11	11	176
17		10	57	77	1	18	18	18	192
18		83	144	450	8	113	113	113	952
19		125	141	192	1	146	146	146	171
20		250	-	-	0	267	-	267	200

nonzero multipliers. The upper bounds quoted are the ones generated before the end of the first pre-processing/Lagrangean relaxation cycle. The other headings are self-explanatory. Problems in groups C, D, and E have 500, 1000 and 2500 vertices, respectively. The algorithm is coded in Fortran and run on a Sun SparcStation 2.

As it can be appreciated from results, the lower bounds we derive are quite sharp, and for 57 of the 60 problems involved, lower bounds equaled optimal SPG solution values. This compares with a figure of 15 out of 60 for Beasley [2] (in same cases we have managed to close gaps of almost 30% between Beasley's lower bounds and optimal solution values). In terms of CPU times, for the 57 problems mentioned above, in a rough comparison with the times quoted by Beasley (based on results in Dongarra [5]), we are always faster (in some cases more than 400 times faster). Our results also appear competitive (using results in Dongarra [5] yet again) with the ones quoted by Chopra, Gorres, and Rao [3], for the same set of test problems. It appears that one algorithm does not dominate the other in terms of performance.

The upper bounds we generated turned out equally sharp. For 55 of the problems considered, optimal solutions were generated before the end of the first pre-processing/Lagrangean relaxation cycle. In addition, for problem E18, to which an optimal solution is unknown, we generated the best known upper bound.

We intend to implement an enumeration scheme, based on the lower bounding approach described in this paper, to attempt to solve to optimality the 3 test problems for which positive duality gaps are reported.

5. Conclusions. Lagrangean relaxation algorithms that incorporate cutting-planes, in the same vein as we describe here, could prove quite attractive to tackle other combinatorial optimization problems. We feel that this holds true despite the fact that,

dual. We allow up to 1000 subgradient iterations to be performed, halving the parameter α that controls the step-size after 40 consecutive iterations without an overall improvement of the lower bound. Whenever a Lagrangean multiplier becomes equal to 0 we drop the associated inequality until it would become violated again. This helps to keep low the number of inequalities being explicitly dualized.

3.1. Preprocessing. We use a number of tests from the literature to attempt to fix variables at preprocessing. Whenever a vertex would have an edge degree of 1 it would either be eliminated from G, if $i \in V \setminus T$, or have its only edge shrunk into a vertex otherwise. A vertex $i \in V \setminus T$ with an edge degree of 2, where both vertices being directly linked to i are also in $V \setminus T$, would be eliminated. This results after the two edges are compressed into a single edge. We have also used the "nearest vertex test" of Beasley [2]. The SST test of Balakrishnan and Patel [1] is used in an attempt to fix edges with both endpoints in T. Finally, we eliminate an edge $(i,j) \in E$, as suggested by Chopra, Gorres and Rao [3], whenever $c_{i,j} \geq \max\{c_{i,k}; c_{j,k}\}$ for $k \in T$.

3.2. Upper Bounds and Run Time Fixing of Variables. We have adapted the algorithm of Takahashi and Matsuyama [14], as refined by Rayward-Smith and Clare [13], to take into account Lagrangean relaxation solutions. We call the upper bounding routine after preprocessing and whenever a new improved lower bound is derived at the Lagrangean relaxation. In the latter case we would temporarily change to 0 the original cost of edges in G that are part of this Lagrangean SST solution (other edge costs remain unaltered). As a result, those edges of cost 0 are made more attractive to be chosen by the Takahashi and Matsuyama algorithm. At the following step (c.f.. Rayward-Smith and Clare [13]), one finds a SST for the subgraph induced by the vertices previously chosen by the Takahashi and Matsuyama algorithm. One then proceeds with pruning this tree of Steiner vertices with edge degree one. At the beginning of this second step we resort again to the original edge costs.

We attempt to fix variables by using the upper bound and the Lagrangean relaxation solution at hand. This is done by computing the associated reduced costs for the variables involved. Whenever the reduced cost of a variable exceeds the difference between the upper bound and the current Lagrangean solution value, one fixes this variable to 0.

3.3. Preprocessing/Lagrangean Relaxation Cycles. We perform the preprocessing and proceed with the Lagrangean relaxation. Whenever, at the end of the subgradient optimization, optimality has not been proven and over 10% of the variables we started the Subgradient Optimization with have been eliminated, we perform the preprocessing/Lagrangean relaxation cycle again. This cycle is repeated until either optimality is proven or the target for variable elimination cannot not be reached. This strategy has proven to be advantageous since the tests for fixing variables that we use tend to propagate their impact into one another. Lucena and Beasley [9] use a similar scheme in the context of a simplex based lower bounding approach.

4. Computational Results. We present computational results for 60 problems proposed by Beasley [2]. In our computations we have explicitly considered inequalities C_1 and C_2 all the time; i.e. even when their Lagrangean multipliers equal to 0, those inequalities are not dropped. We have used the insertion/dropping approach for inequalities C_3, for $|W| \geq 3$, and C_5. We have found it computationally unattractive to use inequalities C_4, at least for $|W| \geq 3$, since no benefits, in terms of improved lower bounds, are noticed. Headings $\max C_3$ and $\max C_5$ in Tables 4.1–4.3 give, at any time, the maximum number of inequalities C_3 (with $|W| \geq 3$) and C_5 with

TABLE 3.1
Effect of varying step size on number of iterations and linear system calls

α	Iterations	Linear System Calls
0.5	25	32
1	17	20
2	8	17
4	5	12
6	4	22
10	4	20
30	5	30

3. Accelerating the Interior Point Method. In [2,5], the step size α used in (2.4) is fixed to $\alpha = 0.5$ throughout their experiments. We observe that it is possible to select α values that are greater than 1. If the potential function $\psi(\mathbf{x})$ is exactly represented by a quadratic function around the given point, then if we move along the Newton direction, we will reach the minimum. So we use $0 \leq \alpha \leq 1$. However, the quadratic function is only a second-order approximation of the original potential function. Consequently, the minimum of the potential function may lie away from the minimum of the quadratic approximation. The step size should be determined in such a way that it reaches a minimum of the potential function on the line of the given descent direction.

We conducted experiments to observe the effect of step size on the number of iterations to solve the 0-1 integer programming representation of the Satisfiability problem described in Kamath *et al.* [2]. We also monitored the number of calls to the HARWELL preconditioned conjugate gradient solver, MA31 [7], to solve linear systems of equations. To show the impact on the iteration count and the number of linear system calls, we considered one problem of 50 variables and 100 clauses. The mean number of literals per clause being 5.059 (see [2] for details). We tabulated the number of iterations and the number of linear system calls for this problem with different α values in Table 3.

The results in Table 3 are typical of the powerful improvement generated by selecting step sizes, α greater than 1 for these classes of integer problems. The iteration numbers decrease while the step size increases, but the iteration numbers increase while the step size is larger than 10. The number of calls to the linear system solver show a similar behavior.

This underscores the need to perform a line search to determine an optimal step size at each iteration in the updating (2.4) . In our algorithm, the step size is determined by a Golden-Section search [6], with the upper bound α_{max}. The value α_{max} is determined empirically. Since the potential function $\psi(\mathbf{x})$ is not known to be unimodular, we can only generate local minima with our present technique.

4. Computational Results. To test our algorithm, we developed a C-language code called **STAR** which embedded a Golden section line search to the algorithm described in [2] to select the minimal value α_{min} in the steepest descent direction $\Delta\mathbf{x}$. To test our algorithm performance, we generated hundreds of random instances of the satisfiability problem by the same method used in [2]. We provide the number of clauses n, the number of variables m and the expected number of literals per clause $E[n(c)]$. Each entry in a random matrix $\hat{\mathbf{A}}$ was set to 1 with probability $\frac{p}{2}$, -1 with

TABLE 4.1
Computational results with STAR: Part 1

Problem			Algorithm	Instances			Iterations		
Vars	Clauses	$E[n(c)]$	Type	Global	local	1-iter	Min	Mean	Max
50	100	5.126	[2] ($\alpha = 0.5$)	89	11	/	1	46.2	530
			$\alpha = 0.5$	97	3	60	1	8.6	83
			α_{min}	95	5	60	1	1.9	9
100	200	5.103	[2] ($\alpha = 0.5$)	82	18	/	1	136.2	1002
			$\alpha = 0.5$	98	2	35	1	22.7	131
			α_{min}	92	8	35	1	3.1	11
200	400	7.029	[2] ($\alpha = 0.5$)	81	19	/	1	412.6	2070
			$\alpha = 0.5$	100	0	59	1	20.4	181
			α_{min}	100	0	59	1	2.8	14
400	800	10.00	[2] ($\alpha = 0.5$)	86	14	/	1	44.2	1320
			$\alpha = 0.5$	100	0	82	0	12.8	289
			α_{min}	100	0	82	0	1.8	17
500	1000	9.996	[2] ($\alpha = 0.5$)	80	20	/	1	119.4	2350
			$\alpha = 0.5$	100	0	82	1	12.3	343
			α_{min}	100	0	82	0	1.8	19

probability $\frac{p}{2}$, and 0 with probability $1 - p$. The value p is determined such that the expected number of nonzero elements per clause is $E[n(c)]$. Empty rows were excluded and also clauses with a single literal.

The results are summarized in Tables 4-4. For each test, we give the number of variables, the number of clauses, the mean number of literals per clause, the number of instances which reach the global minimum, the number of instances which reach a local minimum, the minimum iteration number, the mean iteration number, the maximum iteration number, the number of calls to the HARWELL linear system solver MA31 and the CPU time used. The problems were run on a MIPS machine at the University of Waterloo with the optimization flag -O.

We also compared our results to the results in [2]. Since that work was done using an Alliant FX/80 parallel/vector processing computer, it was possible for the researchers to test larger problems. For instance, problems containing 1000 variables and 2000 clauses were solved. We are able to solve fairly large problems containing 500 variables and 1000 clauses. We felt that it would not be fair to compare running times on different computing environments. However, we have compared the number of iterations obtained using the different algorithms; that is, with and without line search.

We can make the following observations about the test results:

- We were able to reproduce the results in [2] for the case where the step size is fixed; that is, $\alpha = 0.5$. The interior point method produced satisfiable results for the majority of instances. Our implementation achieves more satisfiable truth assignments than [2]. Large step sizes produce local minima for more instances than small step size.
- All problem classes had at least one instance for which the algorithm produced a satisfiable solution in one iteration.
- The problems become hard as the expected number of literals per clause decrease. It is clear that less feasibility is allowed in determining the value of the variable.

of cost 0, attached to it. Denote by $G' = (V', E')$ the resulting expanded graph where $|V'| = |V|+1 = n'$ and let S' be a spanning tree of G'. Taking x as the incidence vector of S', let $x_e = 1$ if $e \in E$ is in set S' and 0 otherwise. Beasley's extended formulation of the SPG consists of a SST for G' subject to $C_1 = \{x_e + x_{i,n+1} \leq 1 : i \in V \setminus T, e \in \delta(i)\}$, where $\delta(i)$ is the set of edges of E with one endpoint in the vertex indexed by i. Constraints C_1 split feasible spanning trees into two halves and, if all edges associated with the artificial vertex $n + 1$ are eliminated (note that those edges would have a combined cost of 0), a Steiner tree for the original graph G would emerge. Beasley relaxed C_1 in a Lagrangean fashion to obtain a valid lower bound for the SPG based on the solution of the resulting unconstrained SST problem.

2.1. Valid Inequalities. Since any Steiner vertex in a SPG solution must have an edge degree of at least 2, a valid inequality that imposes this restriction is given by $C_2 = \{\Sigma(x_e : e \in \delta(i)) + 2x_{i,n+1} \geq 2 : i \in V \setminus T\}$.

A lifting of the subtour elimination constraints would lead to $C_3 = \{x(E(W)) + \Sigma(x_{i,n+1} : i \in W \setminus T) \leq |W| - 1 : W \subseteq V, W \cap T \neq \emptyset\}$, where $E(\cdot)$ denotes the edges with both endpoints in the set of vertices, and $x(\cdot)$ denotes the summation of variables for the edges in the set. Constraints C_3 are an adaptation, within this context, of inequalities introduced independently by Goemans [7], Lucena [8], and Margot, Prodon, and Liebling [10], for another extended formulation of the SPG. In effect, this other extended formulation involves, in addition to binary variables $x_e, e \in E$, binary variables for vertices in $V \setminus T$. Denoting those vertex variables $y_i, i \in V \setminus T$, one has $x_{i,n+1} = 1 - y_i, i \in V \setminus T$. Therefore, variables for the edges of an artificial vertex in Beasley's formulation are no different from variables $y_i, i \in V \setminus T$, for the other formulation (there is a correspondence of one to one between them). Goemans [7] has shown how one can obtain facets defining valid inequalities for the Steiner tree polytope by projecting constraints equivalent to C_3, for the other formulation, onto the space of variables $x_e, e \in E$.

Goemans [7] and Margot, Prodon and Liebling [10] introduced another lifting of the subtour elimination constraints that, in this context, can be expressed as $C_4 = \{x(E(W)) + \Sigma(x_{i,n+1} : i \in W \setminus \{k\}, k \in W) \leq |W| - 1 : W \subset V, W \cap T = \emptyset, k \in W\}$.

Polyhedral results similar to the ones described above for C_3 can also be derived for C_4 (c.f.. Goemans [7]). One should notice that inequalities C_1, introduced by Beasley, are a special case of inequalities C_3 and C_4 for $|W| = 2$.

Finally consider a partition of V into V_1 and V_2 where $V_1 \cap T \neq \emptyset$ and $V_2 \cap T \neq \emptyset$. Let $L(V_1, V_2)$ be the set of edges with one endpoint in V_1 and another endpoint in V_2. Then $C_5 = \{x(L(V_1, V_2)) \leq 1 : V_1, V_2 \subset V, V_1 \cap T \neq \emptyset, V_2 \cap T \neq \emptyset, V_1 \cap V_2 = \emptyset\}$ is valid for the SPG. Inequalities C_5 have been used by Chopra, Gorres and Rao [3] and Lucena and Beasley [9] in their branch and cut algorithms for the SPG. Those inequalities have been shown by Chopra and Rao [4] to be facet defining for the Steiner tree polytope.

3. Lagrangean Relaxation, Upper Bounds and Reduction Tests. We first solve the SST problem for graph G' . Then constraints of types C_2, C_3, C_4, and C_5 that are violated in this SST solution are identified in linear time. Note that this contrasts with the general case where the separation problems associated with constraints C_3, C_4 and C_5 could only be solved in $O(n^4)$ time (that involves solving a number of network flow problems). Violated inequalities, when first detected, are dualized in a Lagrangean fashion. In practical terms, it is as if those constraints have already been dualized (with a multiplier of value 0 being associated with them). We use the Subgradient Method, as suggested by Fisher [11], to optimize the Lagrangean

STEINER PROBLEM IN GRAPHS: LAGRANGEAN RELAXATION AND CUTTING-PLANES

ABILIO LUCENA*

Abstract. We study the use of cutting-planes to strengthen a Lagrangean Relaxation lower bound proposed by Beasley [2] for the Steiner Problem in Graphs. To that order we use strong valid inequalities proposed by us or drawn from the literature. Inequalities are explicitly used only when they first become violated at the solution of a Lagrangean problem. At that stage they are relaxed in a Lagrangean fashion in an attempt to tighten the lower bound. Inequalities are discarded whenever their associated Lagrangean multipliers become equal to zero. Proceeding in this way we are able to deal, in a tractable manner, with valid inequalities that are exponential in number. Computational experiments show that the proposed approach improves significantly the results of Beasley, and is competitive with branch and cut algorithms for the Steiner Problem in Graphs.

1. Introduction. For an undirected graph $G = (V, E)$, with a set of vertices V and a set of edges E, let $T \subseteq V$ ($|T| \geq 2$) be a set of terminal vertices to be connected together. Associated with an edge $e \in E$, there is a cost c_e. A Steiner tree is defined as a tree of G that spans T and, possibly, some vertices in $V \setminus T$. The Steiner Tree Problem (STP) is defined as the problem of finding a least cost Steiner tree. Vertices in $V \setminus T$ that are part of a Steiner tree are denoted Steiner vertices. Whenever costs $c_e \geq 0$, $\forall e \in E$, are used, one can restrict oneself to Steiner trees where no Steiner vertex appears as a leaf (i.e. the Steiner trees are minimal). For this instance, the STP is known as the Steiner Problem in Graphs (SPG).

We consider an extended formulation of the SPG proposed by Beasley [2]. Firstly, an artificial vertex is introduced with edges of cost 0 linking it to all vertices in $V \setminus T$ and to a single (any) vertex in T. Then one proceeds to formulate a Shortest Spanning Tree (SST) problem with additional constraints. Relaxing those additional constraints in a Lagrangean fashion leads to a valid lower bound to the SPG based on the solution of an unconstrained SST problem. In this paper we study the use of cutting-planes, in the context of Lagrangean relaxation, to strengthen the lower bound proposed by Beasley. To that order we use strong valid inequalities proposed by us or drawn from the literature. Inequalities are explicitly used only when they first become violated at the solution of a Lagrangean problem. At that stage they are relaxed in a Lagrangean fashion in an attempt to tighten up the lower bound. Inequalities are discarded whenever their associated Lagrangean multipliers equals to zero. Proceeding in this way we are able to deal, in a tractable manner, with valid inequalities that are exponential in number. This approach, when used within an enumeration tree, presents similarities with branch-and-cut algorithms (e.g. Padberg and Rinaldi [12]). The algorithm of Fisher [6] for the Vehicle Routing Problem appears to have been the first to incorporate cutting-planes, in the context of Lagrangean relaxation, in the way we propose to do here.

Beasley's extended formulation and the strong valid inequalities we use are introduced in section 2. In section 3, a brief description of the algorithm is given. Finally, in section 4 computational results for 60 test problems from the literature are presented.

2. Formulation and Strong Valid Inequalities. Assume that $|V| = n, |E| = m$, and that the vertex indexed by 1 is in T. The artificial vertex to be introduced into G will be indexed by $n+1$ and will have edges $(1, n+1)$ and $(i, n+1), \forall i \in V \setminus T$,

* IRC for Process Systems Engineering, Imperial College, London SW7 2AZ, England

TABLE 4.2
Computational results with STAR: Part 2

Problem			Algorithm	Instances			MA31 calls		
Vars	Clauses	$E[n(c)]$	Type	Global	local	1-iter	Min	Mean	Max
50	100	5.126	[2] ($\alpha = 0.5$)	89	11	/	/	/	/
			$\alpha = 0.5$	97	3	60	4	13.1	127
			α_{min}	95	5	60	4	6.0	25
100	200	5.103	[2] ($\alpha = 0.5$)	82	18	/	/	/	/
			$\alpha = 0.5$	98	2	35	5	29.9	167
			α_{min}	92	8	35	5	10.6	33
200	400	7.029	[2] ($\alpha = 0.5$)	81	19	/	/	/	/
			$\alpha = 0.5$	100	0	59	7	28.4	213
			α_{min}	100	0	59	7	10.9	39
400	800	10.00	[2] ($\alpha = 0.5$)	86	14	/	/	/	/
			$\alpha = 0.5$	100	0	82	0	22.6	324
			α_{min}	100	0	82	0	11.4	49
500	1000	9.996	[2] ($\alpha = 0.5$)	80	20	/	/	/	/
			$\alpha = 0.5$	100	0	82	9	21.4	381
			α_{min}	100	0	82	9	10.3	41

TABLE 4.3
Computational results with STAR: Part 3

Problem			Algorithm	Instances			CPU (seconds)		
Vars	Clauses	$E[n(c)]$	Type	Global	local	1-iter	Min	Mean	Max
50	100	5.126	[2] ($\alpha = 0.5$)	89	11	/	/	/	/
			$\alpha = 0.5$	97	3	60	0.4	2.8	28.3
			α_{min}	95	5	60	0.4	0.9	4.2
100	200	5.103	[2] ($\alpha = 0.5$)	82	18	/	/	/	/
			$\alpha = 0.5$	98	2	35	2.2	29.8	175.9
			α_{min}	92	8	35	2.3	6.6	23.0
200	400	7.029	[2] ($\alpha = 0.5$)	81	19	/	/	/	/
			$\alpha = 0.5$	100	0	59	18.7	168.2	1467.2
			α_{min}	100	0	59	19.0	39.1	170.6
400	800	10.00	[2] ($\alpha = 0.5$)	86	14	/	/	/	/
			$\alpha = 0.5$	100	0	82	0.2	830.9	15778.1
			α_{min}	100	0	82	0.2	249.8	1468.2
500	1000	9.996	[2] ($\alpha = 0.5$)	80	20	/	/	/	/
			$\alpha = 0.5$	100	0	82	313.0	1357.8	31435.0
			α_{min}	100	0	82	316.2	414.1	2422.4

- The interior point algorithm with optimal step size reduces both the number of iterations and the number of calls to the linear system solver significantly.

5. Conclusions. In this note, we presented our computational experience with the interior point algorithm proposed by Kamath *et al.* [2] to solve the Satisfiability problem originally proposed by Cook [1]. For hundreds of randomly generated instances, our implementation produced the satisfiable truth assignments for more instances, and with only $\frac{1}{10}$ to $\frac{1}{20}$ iteration numbers compared to the results reported by Kamath *et al.* [2]. Based on the experimental observations, a significant improvement

to the original Karmarkar algorithm for solving 0-1 integer programs is proposed. This approach adds a line search to the existing interior point method to determine optimal step size rather than using a fixed step size within $(0, 1]$. Computational results show that the modified interior point approach with optimal step size reduced the number of iterations by a factor of 5-15, and the number of calls to the linear system solver by a factor of 2-6, with only about one percent extra time for line search. For a majority of instances of the Satisfiability problem, satisfiability was proven in less than 3 iterations.

REFERENCES

[1] S.A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd ACM Symposium on the Theory of Computing* (1971) 151-158.

[2] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan and M.G.C. Resende, "Computational experience with an interior point algorithm on the Satisfiability problem," *Annals of Operations Research* 25 (1990) 43-58.

[3] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica* 4 (1984) 373-395.

[4] N. Karmarkar, "An interior-point approach to NP-complete problems - Part I," *Contemporary Mathematics* 114 (1990) 297-308.

[5] N. Karmarkar, M.G.C. Resende and K.G. Ramakrishnan, "An interior point algorithm to solve computationally difficult set covering problems," *Mathematical Programming* 52 (1991) 597-618.

[6] D.G. Luenberger *Linear and Nonlinear Programming* (Addison-Wesley, 1984)

[7] N. Munksgaard, "Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients," *TOMS* 6 (1980) 206-219.

[8] S.A. Vavavis, "Quadratic programming is in NP," *Information Processing Letters* 36 (1990) 73-77.

[9] Y. Ye, "On the interior algorithms for nonconvex quadratic programming," to appear *Mathematical Programming* (1992).

SPECIAL ISSUE ON COMPUTATIONAL ASPECTS OF COMBINATORIAL OPTIMIZATION

EDITORIAL

This issue of the COAL Bulletin includes a collection of invited papers that reflects recent research in solving NP-hard combinatorial optimization problems.

In the first paper, Lucena studies the use of cutting-planes to strengthen a Lagrangean relaxation lower bound for the Steiner Problem in Graphs. Computational results on 60 problems show that the approach improves significantly the results of Beasley (1989).

The paper by Kumar and Ananth provides an overview of parallel algorithms for solving combinatorial optimization problems. They discuss parallel depth first and best first search, as well as speedup anomalies and scalability issues.

In the paper by Mitchell and Borchers, an interior point primal-dual cutting plane algorithm for solving the linear ordering problem is given. Promising computational results, on real-world problems, are presented.

Pardalos and Vairaktarakis present instances of maximum clique problems from various applications. The availability of such collections of test problems will facilitate the efforts of comparing performance and correctness of the many proposed algorithms and heuristics for solving the maximum clique problem.

Shi, Vannelli and Vlach describe an improvement in the implementation of Karmarkar's interior point algorithm for integer programming. They apply their implementation on the randomly generated instances of Satisfiability described in Kamath *et al.* (1990) and show that much improvement can be achieved by using a dynamic step size.

We would like to thank the editors for inviting us to prepare this special issue and to thank the contributors for helping us to produce this excellent collection of papers.

Panos M. Pardalos
University of Florida

Mauricio G.C. Resende
AT&T Bell Laboratories

COAL OBJECTIVES

The Committee on Algorithms is involved in computational developments in mathematical programming. There are three major goals: (1) ensuring a suitable basis for comparing algorithms, (2) action as a focal point for computer programs that are available for general calculations and for test problems, and (3) encouraging those who distribute programs to meet certain standards of portability, testing, ease of use and documentation.

BULLETIN OBJECTIVES

The Bulletin's primary objective is to provide a vehicle for the rapid dissemination of new results in computational mathematical programming. To date, our profession has not developed a clear understanding of the issues of how computational tests should be carried out, how the results of these tests should be presented in the literature, or how mathematical programming algorithms should be properly evaluated and compared. These issues will be addressed in the Bulletin.

Mathematical Programming Society

Committee on Algorithms Bulletin

NO. 21
November 1992

JENS CLAUSEN
FAIZ A. AL-KHAYYAL
EDITORS

SPECIAL ISSUE ON COMPUTATIONAL ASPECTS OF COMBINATORIAL OPTIMIZATION

Panos M. Pardalos
Mauricio G. C. Resende

Guest Editors

Editorial 1

Steiner Problem in Graphs: Lagrangean Relaxation and Cutting-Planes . . 2
Abilio Lucena

Parallel Algorithms for Discrete Optimization Problems 8
V. Kumar and Grama Y. Ananth

A Primal-Dual Interior Point Cutting Plane Method for the Linear Ordering Problem 13
John E. Mitchell and Brian Borchers

Test Cases for the Maximum Clique Problem 19
Panos M. Pardalos and George Vairaktarakis

An Improvement on Karmarkar's Algorithm for Integer Programming . . 23
C.-J. Shi, A. Vanneli, and J. Vlach

RØNSHOLT TRYK A/S