Nº65

OPTIMA

Mathematical Programming Society Newsletter

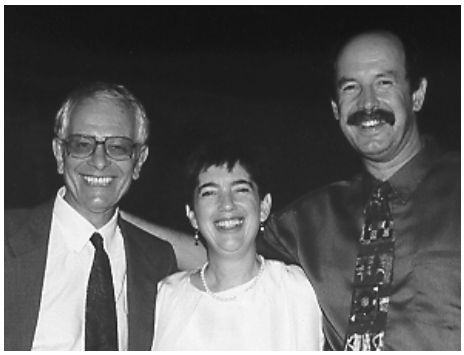
MAY 2001

65

GEORGIA ON MY MIND....

The Seventeenth Symposium on Mathematical Programming – A Great Success

Karen Aardal



SOUTHERN HOSPITALITY! THE HAPPY ORGANIZING TEAM: GEORGE NEMHAUSER, DONNA LLWELLYN, AND MARTIN SAVELSBERGH.

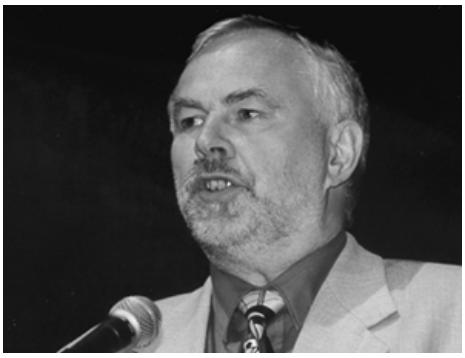
The Symposium was held at the Georgia Institute of Technology, August 7-11, 2000. The main organizers were George Nemhauser, Donna Llwellyn, and Martin Savelsbergh. The atmosphere of the meeting was as warm as the outdoor climate, and the quality of the technical as well as the social program was supreme. There were 1,055 registered participants at the meeting.

Seven plenary talks were given, including a special lecture by Thomas Hales about his proof of the Kepler Conjecture, and the Banquet presentation given by Harold Kuhn. The plenary talks were focusing on the history of some of the key areas of Mathematical Programming, whereas the ten semi-plenary talks were considering more recent developments. About 870 technical talks were given in parallel sessions.

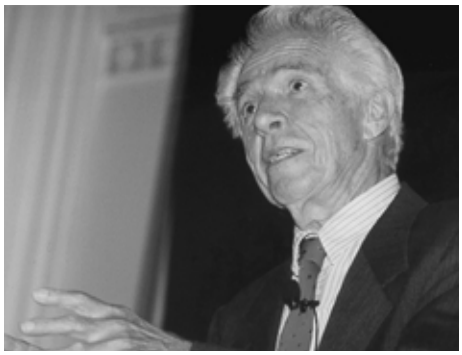
On Sunday evening there was a welcome reception and registration at the Ferst Center for the Arts – the "heart" of the Symposium site. The Symposium was opened on Monday with welcome addresses given by: George Nemhauser; the provost, Michael Thomas; and the MPS chair, Jean-Philippe Vial. During the opening session, the Beale - Orchard-Hays, Dantzig, Fulkerson, and Tucker prizes were awarded as well. The winners and the prize citations can be found on the following pages. Rita Graham contributed with a musical program that was clearly appreciated by the mathematical programmers. I have never seen so many people swinging in their chairs at a conference! On Wednesday there was a fantastic conference banquet. Harold Kuhn gave an entertaining, interesting, and very moving Banquet Presentation that will long remain in our memories.

For the first time in the MPS history, Founders Awards were given to some of our prominent colleagues who have laid the foundation for the many areas of mathematical programming. The ten recipients were: William Davidon, George Dantzig, Lester Ford, David Gale, Ralph Gomory, Alan Hoffman, Harold Kuhn, Harry Markowitz, Philip Wolfe, and Guus Zoutendijk. Lester Ford and David Gale could unfortunately not attend the meeting.

WHICH SESSION TO GO TO NEXT? GEORGE DANTZIG, RALPH GOMORY, AND ELLIS JOHNSON ARE CHECKING THE PROGRAM.

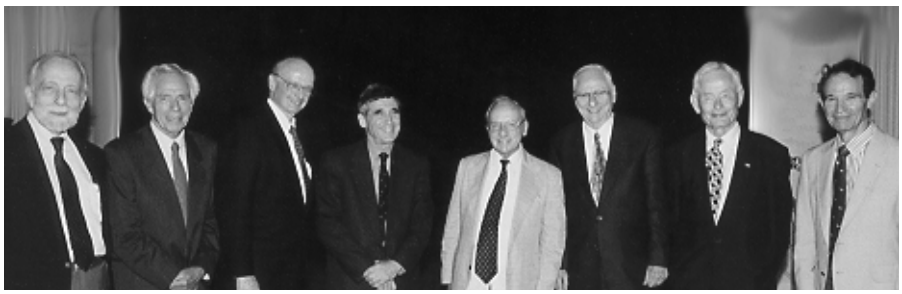


MARTIN GRÖETSCHEL GIVING A PLENARY TALK AT THE OPENING SESSION.



HAROLD KUHN DURING HIS BANQUET PRESENTATION "IN THE RIGHT PLACE AND THE RIGHT TIME."

PAST, PRESENT, AND FUTURE CHAIRMEN OF THE SOCIETY ENJOYING THE BANQUET: GEORGE NEMHAUSER, CHAIR-ELECT BOB BIXBY, JOHN DENNIS, GEORGE DANTZIG, JAN KAREL LENSTRA, AND CURRENT CHAIR JEAN-PHILIPPE VIAL.



EIGHT OF THE FOUNDERS AWARDS RECIPIENTS, FROM LEFT TO RIGHT: PHILIP WOLFE, HAROLD KUHN, HARRY MARKOWITZ, RALPH GOMORY, GEORGE DANTZIG, ALAN HOFFMAN, GUUS ZOUTENDIJK, AND WILLIAM DAVIDON.

THE MPS PRIZES

Listed below are the prize citations for the four MPS prizes (The Beale - Orchard-Hays prize, the Dantzig prize, the Fulkerson prize, and the Tucker prize). Some of the citations have been slightly edited by the OPTIMA Editor.

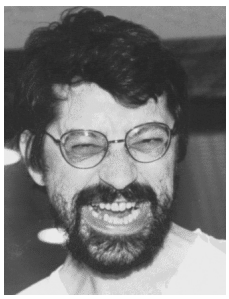
The Beale - Orchard-Hays Prize

The Beale - Orchard-Hays prize – for excellence in computational mathematical programming – was awarded to David Applegate, Robert Bixby, Vašek Chvátal, and William Cook for their paper, "On the Solution of Traveling Salesman Problems," Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung, ICM III (1998), 645-656.

Additional information and source code for the associated CONCORDE software package are available online (http://www.keck.caam.rice.edu/concorde.html – also given in the paper itself).

DAVID APPLEGATE

ROBERT BIXBY



VAŠEK CHVÁTAL

WILLIAM COOK

The Traveling Salesman Problem (TSP) has played a special role in the history of mathematical programming. In addition to its many applications, from manufacturing to scheduling to computational biology, the TSP has long served as the paradigmatic hard combinatorial optimization problem and as a prime first testbed for new ideas in the field, from cutting planes to branch-and-cut to simulated annealing and other metaheuristics.

The award winning paper gives an informative summary of the history of cutting plane based approaches to the TSP and then describes the ideas, both old and new, that the authors have incorporated in their own code, which is now by far the best optimization code for the TSP that has been developed. In its 12 pages (constrained by a page limitation for this special issue of the journal) the paper manages to include a large amount of material at a level such that the non-expert can get a feel for the ideas involved and the expert can understand the key innovations involved in the code.

As with all experimental papers, however, it is the code itself and the results obtained with it that provide the measure of significance. The authors' solution of a 13,509-city instance from the TSPLIB is arguably the most difficult optimization problem that has been solved to date. The 13,509 solution was the culmination of 10 years of work by the group, pushing the envelope of optimization computing in a wide range of areas. The procedure that the authors employed is a model for how to combine mathematical programming techniques with advanced data structures and algorithms, together with parallel computing, to greatly extend the reach of optimization software.

The authors have advanced the state of the art to the point where solving 1,000-city instances is routine. Indeed, in a recent test by Applegate (to gather information on the Beardwood-Halton-Hammersley constant) the authors' code was used to solve 10,000 geometric instances each containing 1,000 points.

The "local cuts" procedure described in the paper (and implemented via a tour de force in their computer code) is an important advance in the way cutting-plane separation algorithms can be viewed. Previously our main route to producing facet-defining inequalities (so useful in practice over the past twenty years) was a template-based approach that required a great deal of problem-specific knowledge, analysis, and algorithms. The new and complementary approach (and its natural extension to general integer programming) provides a general purpose routine that can be specialized to produce facets automatically by a process of linear projection, given only that one has a method for optimizing small instances of the relevant subproblem.

Byproducts of the authors' TSP work have found their way into the leading commercial integer and linear programming codes. A prominent example of this is the strong branching rule (for choosing a branching variable in mixed integer programming branch and bound algorithms) that is now the default choice when solving hard MIP instances.

The authors' source code includes a little over 100,000 lines of C code. This is (to my knowledge) the first time the full source code to a state-of-the-art branch-and-cut algorithm has been made publicly available, and it serves as a model for the development of future efficient implementations. Included with the cutting plane routines are the first publicly-available high-quality implementations of the Lin-Kernighan and Chained-Lin-Kernighan heuristics for finding near-optimal tours, which themselves are of great practical benefit when attempting to get good solutions to instances beyond the range of the optimization routines. All this allows future research to start at the top, rather than having to rebuild a code from the ground up.

The two versions of the accompanying source code (one from 1997 and one from 1999) have had over 1,200 downloads (and the new version continues to obtain about 40 new downloads per week). Links to it have already been added to over 25 web sites (outside of AT&T, Rice, and Rutgers). This amount of activity is unusual (to say the least) for a piece of advanced mathematical software, and a definite measure of its impact.

An honorable mention for the Beale - Orchard-Hays prize was given to the following two papers: Joseph Czyzyk, Michael P. Mesnier, Jorge Moré, The NEOS Server, IEEE Journal on Computational Science and Engineering, 5 (1998), 68-75.

William Gropp, Jorge Moré, Optimization environments and the NEOS Server, in:

Approximation Theory and Optimization, M. D. Buhmann and A. Iserles, eds., pages 167-182, Cambridge University Press, 1997.

The prize jury: M. Ferris, B. Fourer, D. Goldfarb (chair), M. Kojima, M. Saunders.

The George B. Dantzig Prize

The Dantzig prize is jointly administered by the Mathematical Programming Society and the Society for Industrial and Applied Mathematics (SIAM). The prize is awarded to one or more individuals for original research which, by virtue of its originality, breadth and depth, is having a major impact on the field of mathematical programming.

The prize was awarded to Yurii Nesterov. Yurii Nesterov is probably best known for his fundamental contributions to the theory of interior point methods for convex programming, including his introduction of the concept of self-concordance. However, his previous works in convex optimization and in automatic differentiation, though perhaps less widely publicized, were just as creative and authoritative. These accomplishments have justifiably attracted international recognition, and have significantly affected the course of research in optimization theory and algorithms.

The prize jury: W. Cunningham, C. Lemaréchal, S. M. Robinson (chair), L.A. Wolsey.

The D. Ray Fulkerson Prize

The Fulkerson Prize for outstanding papers in the area of discrete mathematics is sponsored jointly by the Mathematical Programming Society and the American Mathematical Society. The jury awarded two prizes at the Symposium.

One award went to Michele Conforti, Gerard Cornuéjols and M. R. Rao for their paper "Decomposition of balanced matrices," Journal Combinatorial Theory, Series B 77 (1999), 292-406.

A balanced matrix is a 0-1 matrix with no submatrix of odd order with precisely two 1's per row and per column. This simply defined class of matrices has been of great interest and is well-studied because packing and covering Linear Programs with balanced constraint matrices have integral solutions. Also, balanced matrices are a natural generalization of bipartite graphs.

The authors give a polynomial time algorithm for recognizing balanced matrices, settling a long-standing open question. The algorithm is based on a deep, beautiful decomposition theorem whose arduous proof combines brilliant insights and meticulous case-checking.

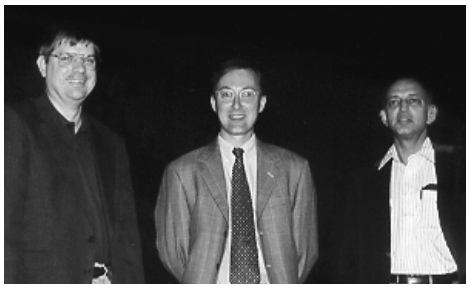
A second award went to Michel Goemans and David Williamson for their paper "Improved approximation algorithms for the maximum-cut and satisfiability problems using semi-definite programming," Journal of the Association for Computing Machinery, 42 (1995), 1115-1145.

This paper gives a much better approximation algorithm for the fundamental maximum-cut problem in graphs and several other optimization problems by an elegant application of semi-definite programming. It is the first paper to apply semi-definite programming in an approximation algorithm with a guaranteed performance bound.

A crucial new element is the extraction of an integer optimal solution from the semi-definite relaxation. The authors' techniques have since found a host of other applications.

The prize jury: W. Cook, R. Graham, R. Kannan (chair).

M.R. RAO, MICHELE CONFORTI, GERARD CORNUÉJOLS, AND RAVI KANNAN (JURY CHAIR)



DAVID WILLIAMSON, MICHEL GOEMANS AND RAVI KANNAN

The A.W. Tucker Prize

The Tucker prize is awarded at each ISMP meeting for the best paper authored by a student.

The committee chose three finalists for the prize, and from among these a winner. The first two finalists are Fabian Chudak from Cornell University, and Kamal Jain from Georgia Tech. Dr. Chudak's paper considers improved approximation algorithms for the uncapacitated facility location problem. Dr. Jain's paper describes a factor-2 approximation algorithm for the generalized Steiner network problem.

The third finalist, and winner of the Tucker prize, is Bertrand Guenin from Carnegie-Mellon University. Dr. Guenin's paper gives a complete characterization of graphs for which a natural LP-relaxation of the MAXCUT problem has all integral vertices. Such graphs were termed "weakly bipartite" by Grötschel and Pulleyblank. Guenin's result verifies a substantial special case of a conjecture by Seymour on ideal clutters dating back to 1977, and represents the first major progress on the resolution of Seymour's conjecture. The paper employs a sophisticated graph-theoretic analysis involving the use of Lehman's results on minimally nonideal matrices, for which Lehman was awarded the MPS Fulkerson prize in 1991. In addition to its purely theoretical interest, the study of ideal matrices has significant applicability in the design of algorithms for partitioning and covering problems that arise in many areas, such as vehicle routing and crew scheduling.

The prize jury: K. Anstreicher (chair), R. Burkhard, D. den Hertog, D. Karger, J. Lee.

Solution of a weighing problem

Cor A.J. Hurkens*
Gerhard J. Woeginger†

1 The seven weights problem

The following problem was published in the Sunday Times on Sunday, 30 June 1963, and a prize of three pounds was offered for the first correct solution:

"On our last expedition to the interior," said the famous explorer, "we came across a tribe who had an interesting kind of Harvest Festival, in which every male member of the tribe had to contribute a levy of grain into the communal tribal store. Their unit of weight was roughly the same as our pound avoirdupois, and each tribesman had to contribute one pound of grain for every year of his age. The contributions were weighed on the tribe's ceremonial scales, using a set of seven ceremonial weights. Each of these weighed an integral number of pounds, and it was an essential part of the ritual that not more than three of them should be used for each weighing, though they need not all be in the same pan.

We were told that if ever a tribesman lived to such an age that his contribution could no longer be weighed by using three weights only, the levy of grain would terminate for ever. And in the previous year, one old man had died only a few months short of attaining the critical age, greatly to the relief of the headman of the tribe. The scientist with our expedition confirmed that the weights had been selected so that the critical age was the maximum possible."

What was the age of the old man when he died? And what were the weights of the seven ceremonial weights?

In a more mathematical formulation, the problem asks for seven positive integers $W_1, \ldots, W_7$ that can represent consecutive integers in the widest possible range $1 \ldots L$. Here an integer n is said to be *represented* by the numbers $W_1, \ldots, W_7$, if there exists an index $1 \le a \le 7$ with $n = W_a$, or if there exist two indices a, b with $1 \le a < b \le 7$ such that $n = \pm W_a \pm W_b$, or if there exist three indices a, b, c with $1 \le a < b < c \le 7$ such that $n = \pm W_a \pm W_b \pm W_c$.

In the solution published in the Sunday Times of 7 July 1963, it was claimed that the best possible range was for L =120, that the weights of the seven ceremonial weights were 1, 3, 7, 12, 42, 75, and 100, and that the old man had died just before his 121st birthday. In this note we show that this claim is not justified: The truth is that the old man had died just before his 123rd birthday. In Section 2, we describe a weight sequence that represents the range 1. . . 122. In Section 3, we argue that a range 1. . . 123 cannot be reached; the argument relies on complete enumeration with the help of a computer program. This program is available on request from the first author (by sending email to wscor@win.tue.nl).

2 Solutions to the problem

When we started our search, we used a primitive program that only searched through a very limited number of feasible solutions. Nevertheless, within two hours of running time this program managed to find a solution for the range 1. . . 122. Table 1 demonstrates that all integers in the range 1. . . 122 indeed can be represented by the seven weights 1, 3, 7, 12, 43, 76, and 102.

We then completely settled the problem by performing an implicit complete enumeration with the help of a computer program. For this we needed a more sophisticated approach, which is described in detail in Section 3. In Table 2, we list in lexicographically increasing order all possible sets of seven weights that can represent ranges $1 \ldots L$ with $L \ge 120$.

3 Implicit enumeration

We have implemented an enumeration scheme to find the best possible sequence of weights W_i, where $W_1 \le W_2 \le W_3 \le W_4 \le W_5 \le W_6 \le W_7$. Our enumeration scheme searches through the weight sequences in lexicographically increasing order. A naive approach that simply enumerates *all* possible sequences will fail since the number of combinations to be inspected is fairly large: Even if we could assume a priori that $W_7 \le 123$ (and there is absolutely no reason for assuming this), there still would be roughly 10^{11} sequences to consider; and for each single one of these 10^{11} sequences, we had to check whether it represents the full range 1 . . . 122 of integers.

*wscor@win.tue.nl; Department of Mathamatics and Computing Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands.
†gwoegi@opt.math.tu-graz.ac.at; Institut für Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria. Supported by the START program Y43-MAT of the Austrian Ministry of Science.

1 = 1	26 = –76 + 102	51 = 1 + 7 + 43	76 = 76	101 = –1 + 102
2 = –1 + 3	27 = 1 – 76 + 102	52 = –3 + 12 + 43	77 = 1 + 76	102 = 102
3 = 3	28 = – 3 – 12 + 43	53 = 3 + 7 + 43	78 = –1 + 3 + 76	103 = 1 + 102
4 = 1 + 3	29 = 3 – 76 + 102	54 = –1 + 12 + 43	79 = 3 + 76	104 = –1 + 3 + 102
5 = –7 + 12	30 = –1 – 12 + 43	55 = 12 + 43	80 = 1 + 3 + 76	105 = 3 + 102
6 = –1 + 7	31 = –12 + 43	56 = 1 + 12 + 43	81 = –7 + 12 + 76	106 = 1 + 3 + 102
7 = 7	32 = 1 – 12 + 43	57 = –7 – 12 + 76	82 = –1 + 7 + 76	107 = –7 + 12 + 102
8 = 1 + 7	33 = –43 + 76	58 = –1 – 43 + 102	83 = 7 + 76	108 = –1 + 7 + 102
9 = –3 + 12	34 = 1 – 43 + 76	59 = –43 + 102	84 = 1 + 7 + 76	109 = 7 + 102
10 = 3 + 7	35 = –1 – 7 + 43	60 = 1 – 43 + 102	85 = –3 + 12 + 76	110 =1 + 7 + 102
11 = –1 + 12	36 = –7 + 43	61 = –3 – 12 + 76	86 = 3 + 7 + 76	111 = –3 + 12 + 102
12 = 12	37 = 1 – 7 + 43	62 = 3 – 43 + 102	87 = –1 + 12 + 76	112 = 3 + 7 + 102
13 = 1 + 12	38 = 7 – 12 + 43	63 = –1 – 12 + 76	88 = 12 + 76	113 = –1 + 12 + 102
14 = –1 + 3 + 12	39 = –1 – 3 + 43	64 = –12 + 76	89 = 1 + 12 + 76	114 =12 + 102
15 = 3 + 12	40 = –3 + 43	65 = 1 – 12 + 76	90 = –12 + 102	115 = 1 + 12 + 102
16 = 1 + 3 + 12	41 = 1 – 3 + 43	66 = –3 – 7 + 76	91 = 1 – 12 + 102	116 = –3 + 43 + 76
17 = 43 + 76 – 102	42 = –1 + 43	67 = 3 – 12 + 76	92 = –3 – 7 + 102	117 = 3 + 12 + 102
18 = –1 + 7 + 12	43 = 43	68 = –1 – 7 + 76	93 = 3 – 12 + 102	118 = –1 + 43 + 76
19 = 7 + 12	44 = 1 + 43	69 = –7 + 76	94 = –1 – 7 + 102	119 = 43 + 76
20 = 1 + 7 + 12	45 = –1 + 3 + 43	70 = 1 – 7 + 76	95 = –7 + 102	120 = 1 + 43 + 76
21 = –12 – 43 + 76	46 = 3 + 43	71 = 7 – 12 + 76	96 = 1 – 7 + 102	121 = 7 + 12 + 102
22 = 3 + 7 + 12	47 = 1 + 3 + 43	72 = –1 – 3 + 76	97 = 7 – 12 + 102	122 = 3 + 43 + 76
23 = –3 – 76 + 102	48 = –7 + 12 + 43	73 = 3 + 76	98 = –1 – 3 + 102	
24 = –7 – 12 + 43	49 = –1 + 7 + 43	74 = 1 – 3 + 76	99 = –3 + 102	
25 = –1 – 76 + 102	50 = 7 + 43	75 = –1 + 76	100 = 1 – 3 + 102	

Table 1: How to represent all the integers in the range 1. . . 122 by the seven weights 1, 3, 7, 12, 43, 76, and 102.

Hence, we implemented several tricks that helped to considerably cut down the search tree. These tricks investigate potential extensions of partial solutions. We say that a partial solution $W_1 \le \cdots \le W_\lambda$ (with $1 \le \lambda \le 6$) is *interesting*, if there is still some possibility to extend it by numbers $W_{\lambda+1}, \ldots, W_7$, to a sequence that represents all the numbers in the range $1 \ldots \theta$. The value θ is a threshold that will be set to some value ≥ 117; the choice of this threshold is somewhat arbitrary, but the analysis below is general. In order to find good bounds on the number of distinct representable values in the range $1 \ldots \theta$, we compute and maintain the following sets and numbers for every partial solution $W_1 \le \cdots \le W_\lambda$.

- $\lambda \le 6$ is the number of currently fixed weights.
- $\mathcal{M}(k, \lambda)$ with $1 \le k \le 3$ is the set of all integers that can be represented by using at most k numbers from $W_1, \ldots W_\lambda$. This set includes 0 and negative numbers.
- W^- is a (tentative) lower bound on the numbers $W_{\lambda+1}, \ldots, W_7$.

From now on we will write $\#(X)$ to denote the cardinality of a set X. Note that the cardinalities of the sets $\mathbb{N} \cap \mathcal{M}(1, \lambda)$, $\mathbb{N} \cap \mathcal{M}(2, \lambda)$, and $\mathbb{N} \cap \mathcal{M}(3, \lambda)$ can be bounded from above by the values λ, λ^2, and $\lambda^2 + 4\binom{\lambda}{3}$, respectively. Similarly, the cardinalities of the sets $\mathcal{M}(1, \lambda)$, $\mathcal{M}(2, \lambda)$, and $\mathcal{M}(3, \lambda)$ can be bounded from above by $2\lambda + 1$, $2\lambda^2 + 1$ and $2\lambda^2 + 8\binom{\lambda}{3} + 1$, respectively. We will use these bounds many times in our arguments.

λ	1	2	3	4	5	6	7
λ^2	1	4	9	16	25	36	49
$\lambda^2 + 4\binom{\lambda}{3}$	1	4	13	32	65	116	189

3.1 Locally bounding the depth of the search tree

The following expression $A(\lambda, W^-, \theta)$ is an upper bound on the number of distinct positive integers in $1 \ldots \theta$ that can be represented from any extension of the current partial solution. For ease of exposition, any number that can be chosen from the set $\mathcal{M}(\cdot, \cdot)$ is denoted by M_i, and any weights yet to be fixed are denoted by $W_j \le W_k \le W_l$ with $j < k < l$.

$A(\lambda, W^-, \theta) =$
$\#([1, \theta] \cap \mathcal{M}(3, \lambda))$ /* combinations without any W_j */

$+\binom{7-\lambda}{1} \#((-\infty, \theta - W^-] \cap \mathcal{M}(2, \lambda))$
/* combinations abs($W_j \pm M_i$) */

$+\binom{7-\lambda}{2} \#(\mathcal{M}(1, \lambda))$ /* combinations abs($W_j - W_k \pm M_i$) */

$+\binom{7-\lambda}{2} \#((-\infty, \theta - 2W^-] \cap \mathcal{M}(1, \lambda))$ /* combinations abs($W_j + W_k \pm M_i$) */

W_1	W_2	W_3	W_4	W_5	W_6	W_7	Range
1	2	12	17	33	74	99	1...120
1	3	7	12	42	75	100	1...120
1	3	7	12	43	76	102	1...122
1	3	9	14	40	70	103	1...120
1	3	10	15	38	72	103	1...121
2	4	7	22	35	78	90	1...121
2	5	6	15	43	74	103	1...120
2	6	13	23	33	83	84	1...120
3	5	15	16	33	75	99	1...120

Table 2: All possible solutions for ranges $1 \ldots L$ with $L \ge 120$

$+4\binom{7-\lambda}{3}$ /* combinations abs($\pm W_j \pm W_k \pm W_l$) */

$$-\begin{cases} \binom{7-\lambda}{3} & \text{if } 3W^- > \theta \\ 0 & \text{otherwise} \end{cases}$$ /*compensating for too high triples $W_j + W_k + W_l$ */

$$-\begin{cases} 2\binom{7-\lambda}{3} & \text{if } W^- > \theta \\ 0 & \text{otherwise} \end{cases}$$ /*compensating for $-W_j + W_k + W_l$ and $W_j - W_k + W_l$ */

E.g., the fifth additive term $4\binom{7-\lambda}{3}$ is justified by the observation that at most 4 of the 8 combinations $\pm W_j \pm W_k \pm W_l$ can be positive. Note that the function $A(\lambda, W^-, \theta)$ is non-increasing in parameter W^-, and that $W^- \ge W_\lambda$. Hence, if $A(\lambda, W_\lambda, \theta) < \theta$ holds, then the considered partial solution cannot be interesting. In this case, we skip all extensions of this partial solution.

3.2 Locally bounding the width of the search tree

If a partial solution survives the above feasibility test and satisfies $A(\lambda, W_\lambda, \theta) \ge \theta$, then our goal is to compute lower and upper bounds on $W_{\lambda+1}$, the value of the next weight to be fixed. Since we generate the weight sequences in lexicographically increasing order, the value W_λ yields a trivial lower bound for $W_{\lambda+1}$. Now let us look for upper bounds. Intuitively it is clear that if we set the weight $W_{\lambda+1}$ too high, then we will not be able to represent all the 'small' integers below $W_{\lambda+1}$. To make this idea precise, we compute a crude upper bound $B(\lambda, \theta)$ on the number of integers in the range $1 \ldots \min\{\theta, W_{\lambda+1} - 1\}$ that can be represented by any extension of the considered partial solution:

$B(\lambda, \theta) =$
$\#([1, \theta] \cap \mathcal{M}(3, \lambda))$ /* combinations without any W_j */

$+\binom{7-\lambda}{1}$ #([1, ∞) ∩ $\mathcal{M}$ (2, λ))
/* combinations $W_j - M_i$ */

$+\binom{7-\lambda}{2}$ #(M (1, λ)) /* combinations abs ($W_j - W_k \pm M_i$) */

$+\binom{7-\lambda}{3}$ /* combinations abs($W_j + W_k - W_l$) */

(1) First we discuss the case where $B(\lambda, \theta) < \theta$ holds. Here we use the upper bound $W_{\lambda+1} \le B(\lambda, \theta) + 1$. Note that we can bound $B(\lambda, \theta)$ from above by $\lambda^2 + 4\binom{\lambda}{3} + \binom{7-\lambda}{1}\lambda^2 + \binom{7-\lambda}{2}(2\lambda + 1) + \binom{7-\lambda}{3}$. Here $\lambda^2 + 4\binom{\lambda}{3}$ is an upper bound on #([1, θ] ∩ $\mathcal{M}$(3, λ)), and $\binom{7-\lambda}{1}\lambda^2$ is an upper bound on $\binom{7-\lambda}{1}$ #([1, ∞) ∩ $\mathcal{M}$ (2, λ)) and so on. This yields the following a priori upper bounds on $W_{\lambda+1}$:

λ	0	1	2	3	4	5	6
$B(\lambda, \theta) \le$	56	72	84	95	108	126	152
$W_{\lambda+1} \le$	57	73	85	96	109	___	___

In fact if $B(\lambda, \theta) < \theta$, we may compute an even better upper bound on $W_{\lambda+1}$ by excluding the very high values in $\mathcal{M}(3, \lambda)$. Define

$$x^* = \max\{x \mid x \le \theta \text{ and } B(\lambda, x) \ge x\}.$$

Then the inequality $W_{\lambda+1} \le B(\lambda, x^* + 1) + 1$ yields the improved upper bound on $W_{\lambda+1}$.

(2) Next we discuss the remaining case where $B(\lambda, \theta) \ge \theta$. Here we define

$$y^* = \min\{y \mid A(\lambda, y, \theta) < \theta\}$$

By the definition of function A in the preceding section, we get that for $W^- \ge y^*$ there is no feasible extension of the considered partial solution. Hence, in this second case we arrive at the upper bound $W_{\lambda+1} \le y^* - 1$.

It remains to be shown that the above defined value y^* indeed exists, i.e., that the minimum in the right hand side exists (which is not at all obvious). By examining the function A, one observes that $A(\lambda, \infty, \theta)$ is bounded from above by $\lambda^2 + 4\binom{\lambda}{3} + (2\lambda + 1)\binom{7-\lambda}{2} + \binom{7-\lambda}{3}$: The right hand side of the definition is the sum of seven terms. The first term #([1, θ] ∩ $\mathcal{M}$(3, λ)) is bounded from above by $\lambda^2 + 4\binom{\lambda}{3}$. The second and the fourth term disappear for sufficiently large values of W^-. The third term $\binom{7-\lambda}{2}$ #$\mathcal{M}$(1, λ)) is bounded from above by $\binom{7-\lambda}{2}(2\lambda + 1)$. The sum of the last three terms equals $\binom{7-\lambda}{3}$ for sufficiently large W^-. Altogether, this yields the following table.

λ	1	2	3	4	5	6	7
$A(\lambda, \infty, \theta) \le$	66	64	59	60	76	116	189

Since $\lambda \le 6$ and since $\theta > 116$, the value y^* indeed exists. Finally, we remark that the value y^* can be computed quickly by bisection search.

To summarize, for every partial solution we first determine the value $B(\lambda, \theta)$. Depending on whether this value is $< \theta$ or $\ge \theta$, we compute an upper bound that is based on x^* or y^*, respectively. Then we only search extensions of the partial solution for which $W_{\lambda+1}$ takes values between W_λ and this upper bound.

3.3 Fathoming a node by looking ahead

Starting from an empty sequence of weights, we recursively compute and extend partial solutions. The bounds described above are used to limit the number of combinations that have to be considered. It is computationally cheap to evaluate the functions A and B, since we compute and maintain all necessary auxiliary values and sets on the fly.

Once five weights have been fixed (i.e., once we are down to λ = 5), it is possible to efficiently determine whether the partial solution is interesting. By a slight adaptation of routine A, we can compute even sharper bounds on the value W_7. This allows us to skip several additional values of W_6. Once six weights have been fixed, it is an easy job to find the interesting values for W_7. At this stage, almost every partial solution indeed leads to a sequence which represents each integer in 1 . . . θ.

3.4 Remarks on the computer program

The program has been coded in C and took 475 minutes of CPU time on a 450 MHz PC running under Linux. The final search tree has approximately 68 million leaf nodes. The last 5-tuple (W_1, W_2, W_3, W_4, W_5) that was investigated is (57, 73, 84, 91, 95). The highest values considered for the seven weights are 57, 73, 85, 96, 109, 110, and 103, respectively. There exist 116,315,910 5-tuples ($W_1 \le W_2 \le W_3 \le W_4 \le W_5$), with W_i below the respective upper bounds, so at least half of them have been fathomed by our bounding procedures. This shows that the search tree has been kept under control, as far as the higher weights are concerned.

We have not been able to cut down the range for the first weights W_1 and W_2. The reader may want to observe from the entries in Table 2 that good solutions are possible without using the values 1 and 2 at all. Nevertheless, one would expect some argument showing that the partial solutions in which, say, $W_1 \ge 10$ need not be considered. Such a result would speed up the algorithm considerably. The highest upper bounds on W_6 and W_7 (observed when the first five weights have already been fixed) are 137 and 262, respectively. This indicates that it will be rather difficult to bound the ranges of these variables a priori.

We conclude that only the application of the various bounds enabled us to implicitly enumerate all solutions.

4 Concluding remarks

Table 2 is the result of the search setting the threshold θ = 120. It therefore contains all possible solutions from which a consecutive range 1 . . . 120 can be represented.

This type of weighing problem should yield nice challenges for the constraint programming community. After all, we have invested a lot of work and a lot of thinking into the solution of this problem, and it is not clear whether the generic constraint programming approaches (where one only needs to formulate a problem in some meta-language and then all the optimization and all the searching are performed automatically by the underlying software) can compete against human specialization.

Acknowledgement

We thank Clive Tooth for stating this weighing problem on his web page at (http://www.pisquaredoversix.force9.co.uk/3From7.htm).

Solving Optimization Problems on Computational Grids

Stephen J. Wright*

November 21, 2000

1 Multiprocessors and Computational Grids

Multiprocessor computing platforms, which have become available more and more widely since the mid-1980s, are now heavily used by organizations that need to solve very demanding computational problems. There has also been a great deal of research on computational techniques that are suited to these platforms, and on the software infrastructure needed to compile and run programs on them. Parallel computing is now central to the culture of many research communities, in such areas as meteorology, computational physics and chemistry, and cryptography. Nevertheless, fundamental research in numerical techniques for these platforms remains a major topic of investigation in numerical PDE and numerical linear algebra.

The nature of parallel platforms has evolved rapidly during the past 15 years. The Eighties saw a profusion of manufacturers (Intel, Denelcor, Alliant, Thinking Machines, Convex, Encore, Sequent, among others) with a corresponding proliferation of architectural features: hypercube, mesh, and ring topologies; shared and distributed memories; memory hierarchies of different types; butterfly switches; and global buses. Compilers and runtime support tools were machine specific and idiosyncratic. Argonne's Advanced Computing Research Facility kept a zoo of these machines during the late 1980s, allowing free access to many researchers in the United States and giving many of us our first taste of this brave new world.

By the early 1990s, the situation had started to change and stabilize. Most of the vendors went out of business, and their machines gradually were turned off – one processor at a time, in some cases. Architectures gravitated toward two easily understood paradigms that prevail today. One paradigm is the shared-memory model typified by the SGI Origin series and by computers manufactured by Sun and Hewlett-Packard. The other paradigm, typified by the IBM SP series, can be viewed roughly as a uniform collection of processors, each with its own memory and all able to pass messages to one another at a rate roughly independent of the locations of the two processors involved. On the software side, the advent of software tools such as p4, MPI, and PVM allowed users to write code that could be compiled and executed without alteration on the machines of different manufacturers, as well as on networks of workstations.

The optimization community was quick to take advantage of parallel computers. In designing optimization algorithms for these machines, it was best in some cases to exploit parallelism at a lower level (that is, at the level of the linear algebra or the function/derivative evaluations) and leave the control flow of the optimization algorithm essentially unchanged. Other cases required a complete rethinking of the algorithms, to allow simultaneous exploration of different regions of the solution space, different parts of the branch-and-bound tree, or different candidates for the next iterate. Novel parallel approaches were developed for global optimization, network optimization, and direct-search methods for nonlinear optimization. Activity was particularly widespread in parallel branch-and-bound approaches for various problems in combinatorial and network optimization, as a brief Web search can attest.

As the cost of personal computers and low-end workstations has continued to fall, while the speed and capacity of processors and networks have increased dramatically, "cluster" platforms have become popular in many settings. Though the software infrastructure has yet to mature, clusters have made supercomputing inexpensive and accessible to an even wider audience.

A somewhat different type of parallel computing platform known as a *computational grid* (alternatively, *metacomputer*) has arisen in comparatively recent times [1]. Broadly speaking, this term refers not to a multiprocessor with identical processing nodes but rather to a heterogeneous collection of devices that are widely distributed, possibly around the globe. The advantage of such platforms is obvious: they have the potential to deliver enormous computing power. (A particular type of grid, one made up of unused computer cycles of workstations on a number of campuses, has the additional advantage of costing essentially nothing.) Just as obviously, however, the complexity of grids makes them very difficult to use. The software infrastructure and the applications programs that run on them must be able to handle the following features:

> heterogeneity of the various processors in the grid;
>
> the dynamic nature of the platform (the pool of processors available to the user may grow and shrink during the computation);
>
> the possibility that a processor performing part of the computation may disappear without warning;
>
> latency (that is, time for communications between the processors) that is highly variable but often slow.

In many applications, however, the potential power and/or low cost of computational grids make the effort of meeting these challenges worthwhile. The Condor team, headed by Miron Livny at the University of Wisconsin, were among the pioneers in providing infrastructure for grid computations. The Condor system can be used to assemble a parallel platform from workstations, PC clusters, and multiprocessors and can be configured to use only "free" cycles on these machines, sharing them with their respective owners and other users. More recently, the Globus project has developed technologies to support computations on geographically distributed platforms consisting of high-end computers, storage and visualization devices, and other scientific instruments.

In 1997, we started the metaneos project as a collaborative effort between optimization specialists and the Condor and Globus groups. Our aim was to address complex, difficult optimization problems in several areas, designing and implementing the algorithms and the software infrastructure needed to solve these problems on computational grids. A coordinated effort on both the optimization and the computer science sides was essential. The existing Condor and Globus tools were inadequate for direct use as a base for programming optimization algorithms, whose control structures are inevitably more complex than those required for task-farming applications. Many existing parallel algorithms for optimization were "not parallel enough" to exploit the full power of typical grid platforms. Moreover, they were often "not asynchronous enough," in that they required too much communication between tasks to execute efficiently on platforms with the heterogeneity and communications latency properties of our target platforms. A further challenge was that, in contrast to other grid applications, the computational resources required to solve an optimiza-

*Mathematics and Computer Science Division, Argonne National Laboratory, and Computer Science Department, University of Chicago.

tion problem often cannot be predicted with much confidence, making it difficult to assemble and utilize these resources effectively.

This article describes some of the results we have obtained during the first three years of the metaneos project. Our efforts have led to development of the runtime support library MW, for implementing algorithms with master-worker control structure on Condor platforms. This work is discussed below, along with our work on algorithms and codes for integer linear programming, the quadratic assignment problem, and stochastic linear programming. Other metaneos work, not discussed below, includes work in global optimization, integer nonlinear programming, and verification of solution quality for stochastic programming.

2 Condor, Globus, and the MW Framework

The Condor system [2, 3] had its origins at the University of Wisconsin in the 1980s. It focuses on collections of computing resources, known as *Condor pools*, that are distributively owned. To understand the implications of "distributed ownership," consider a typical machine in a pool: a workstation on the desk of a researcher. The Condor system provides a means by which other users (not known to the machine's owner) can exploit some of the unused cycles on the machine, which otherwise would sit idle most of the time. The owner maintains control over the access rights of Condor to his machine, specifying the hours in which Condor is allowed to schedule processes on the machine and the conditions under which Condor must terminate any process it is running when the owner starts a process of his own. Whenever Condor needs to terminate a process under these conditions, it migrates the process to another machine in the pool, guaranteeing eventual completion.

When a user submits a process, the Condor system finds a machine in the pool that matches the software and hardware requirements of the user. Condor executes the user's process on this machine, trapping any system calls made by the process (such as input/output operations) and referring them back to the submitting machine. In this way, Condor preserves much of the submitting machine's environment on the execution machine. Users can submit a large number of processes to the pool at once. Since each such process maintains contact with the submitting machine, this feature of Condor opens up the possibility of parallel processing. Condor provides an *opportunistic* environment, one that can make use of whatever resources currently are available in its pool. This set of resources grows and shrinks dynamically during execution of the user's job, and his algorithm should be able to exploit this situation.

The Globus Toolkit [4] is a set of components that can be used to develop applications or programming tools for computational grids. Currently, the Toolkit contains tools for resource allocation management and resource discovery across a grid, security and authentication, data movement, message-passing communication, and monitoring of grid components. The main use of Globus within the metaneos project has been at a level below Condor. By a Globus mechanism known as *glide-in*, a user can add machines at a remote location into the Condor pool on a temporary basis, making them accessible only to his own processes. In this way, a user can marshal a large and powerful set of resources over multiple sites, some or all of them dedicated exclusively to his job.

MW is a software framework that facilitates implementation of algorithms of master-worker type on computational grids. It was developed as part of the metaneos project by Condor team members Mike Yoder and Sanjeev Kulkarni in collaboration with optimization specialists Jeff Linderoth and Jean-Pierre Goux [5, 6]. MW takes the form of a set of C++ abstract classes, which the user implements to perform the particular operations associated with his algorithm and problem class. There are just ten virtual functions, grouped into the following three fundamental base classes:

- **MWDriver** contains four functions that obtain initial user information, set up the initial set of tasks, pack the data required to initialize each worker processor as it becomes available, and act on the results that are returned to the master when a task is completed.
- **MWWorker** contains two functions, to unpack the initialization data for the worker and to execute a task sent by the master.
- **MWTask** contains four functions to pack and unpack the data defining a single task and to pack and unpack the results associated with that task.

MW also contains functions that monitor performance of the grid and gather various statistics about the run.

Internally, MW works by managing a list of workers and a list of tasks. The resource management mechanisms of the underlying grid are used to obtain new workers for the list and provide information about each worker. The information can be used to order the worker list so that the most suitable workers (e.g., the fastest machines) are at the head of the list and hence are the first to receive tasks. Similarly, the task list can be ordered by a user-defined key to ensure that the most important tasks are performed first. Scheduling of tasks to workers then becomes simple: The first task on the list is assigned to the first available worker. New tasks are added to the list by the master process in response to results received from completion of an earlier task.

MW is currently implemented on two slightly different grid platforms. The first uses Condor's version of the PVM (parallel virtual machine) protocol, while the second uses the remote I/O features of Condor to allow master and workers to communicate via series of shared files. In addition, MW provides a "bottom-level" interface that allows it to be implemented in other grid computing toolkits.

3 Integer Programming

Consider the linear mixed integer programming problem

$$\min c^T x \text{ subject to } Ax \le b,\ l \le x \le u,$$

$$x_i \in Z, \text{ for all } i \in I,$$

where x is a vector of length n, Z represents the integers, and $I \subset \{1,2,...,n\}$. Parallel algorithms and frameworks for this problem have been investigated by a number of authors in recent times. The approaches described in [7, 8, 9, 10] implement enhancements of the branch-and-bound procedure, in which the work of exploring the branch-and-bound tree is distributed among a fixed number of processors. These approaches are differentiated by their use of virtual-shared-memory vs. message-passing models, their load balancing procedures, their choice of branching rules, and their use of cuts.

By contrast, the FATCOP code of Chen, Ferris, and Linderoth [11] uses the MW framework to greedily use whatever computational resources become available from the Condor pool. The algorithm implemented in FATCOP (the name is a loose acronym for "fault-tolerant Condor PVM") is an enhanced branch-and-bound procedure that utilizes (globally valid) cuts, pseudocosts for branching, preprocessing at nodes within the branch-and-bound tree, and heuristics to identify integer feasible solutions rapidly.

In FATCOP's master-worker algorithm, each task consists of exploration of a subtree of the branch-and-bound tree, not just evaluation of a single node of the tree. Given a root node for the subtree, and other information such as the global cut and pseudocost pools, the task executes for a given amount of time, making its own branching decisions and accumulating its

own collection of cuts and pseudocosts. It may also perform a "diving" heuristic from its root node to seek a new integer feasible solution. When the task is complete, it returns to the master a stack representing the unexplored portions of its subtree. (Depth-first search is used to limit the size of this stack.) The task also sends back any new cut and pseudocost information it generated, which is added to the master's global cut and pseudocost pools.

By processing a subtree rather than a single node, FATCOP increases the granularity of the task and improves utilization of the computational power of each worker. The time for which a processor is sitting idle while waiting for the task information to be sent to and from the master, and for the master to process its results and assign it a new task, is generally small relative to the computation time.

The master process is responsible for maintaining the task pool, as well as the pools of cuts and pseudocosts. It recognizes new workers as they join the computation pool, and sends them copies of the problem data together with the current cut and pseudocost pools. Moreover, it sends tasks to these workers and processes the results of these tasks by updating its pools and possibly its incumbent solution. Another important function of the master is to detect when a machine has disappeared from the worker pool.

Table 1: Performance of FATCOP on gesa2_o

	$\bar{P}$	Nodes	Time
minumum	43.2	6207993	6951
average	62.5	8214031	10074
maximum	86.3	9693518	13198

In this case, the task that was occupying that machine is lost, and the master must assign it to another worker. (This is the "fault tolerant" feature that makes FATCOP fat!) On long computations, the master process "checkpoints" by writing out the current state of computation to disk. By doing so, it can restart the computation at the latest checkpoint after a crash of the master processor.

To illustrate FATCOP's performance, we consider the solution of the problem gesa2_o from the MIPLIB test set. This problem arises in an electricity generation application in the Balearic Islands. FATCOP was run ten times on the Condor pool at the University of Wisconsin. Because of the dynamic computational environment — the size and composition of the pool of available workers and communication times on the network varied between runs and during each run — the search pattern followed by FATCOP was quite different in each instance, and different from what one would obtain from a serial implementation of the same approach. However, using the statistical features of MW, we can find the average number of workers used during each run, defined as

$$\bar{P} = \sum_{k=1}^{\infty} k\tau_k / T,$$

where τ_k is the total time during which the run had control of k processors, and T is the wall clock time for the run. The minimum, maximum, and mean values of $\bar{P}$ over the ten runs are shown in Table 1. This table also shows statistics for the number of nodes evaluated by FATCOP, and the wall clock times.

Figure 1: Number of Workers during FATCOP on gesa2_o

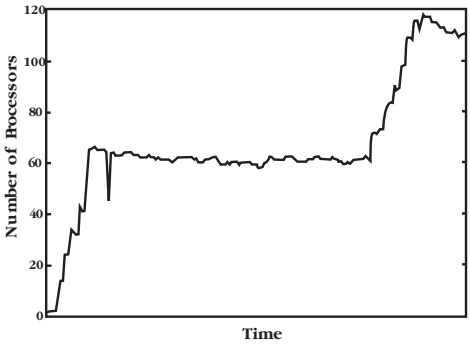


Figure 1 profiles the size of the worker pool during a particular run. Note the sharp dip, which occurred when a set of machines participated in a daily backup procedure, and the gradual buildup during the end of the run, which occurred in the late afternoon when more machines became available as their owners went home.

For a detailed performance analysis of FATCOP, see [11].

A separate but related activity involved solving to optimality the well-known *seymour* problem from the MIPLIB library of integer programs. This well-known integer-programming problem, posed by Paul Seymour, arises in a new proof [12] of the famous Four-Color Theorem, which states that any map can be colored using four colors in such a way that regions sharing a boundary segment receive different colors. The *seymour* problem is to find the smallest set of configurations such that the Four-Color Theorem is true if none of these configurations can exist in a minimal counterexample. Although Seymour claimed to have found a solution with objective value 423, nobody (including Seymour himself) had been able either to reproduce this solution, or prove a strong lower bound on the optimal value. There was some skepticism in the integer programming community as to whether this was indeed the optimal value.

In July 2000, a team of metaneos researchers found solutions with the value 423, and proved its optimality. Gabor Pataki, Stefan Schmieta, and Sebastian Ceria at Columbia used preprocessing, disjunctive cuts and branch-and-bound to break down the problem into a list of 256 integer programs. Michael Ferris at Wisconsin and Jeff Linderoth at Argonne joined the Columbia group in working through this list. The problems were solved separately with the help of the Condor system, using the integer programming packages CPLEX and XPRESS-MP. About 9,000 hours of CPU time was needed, the vast majority of it spent in proving optimality.

4 Quadratic Assignment Problem

The quadratic assignment problem (QAP) is a problem in location theory that has proved to be among the most difficult combinatorial optimization problems to solve in practice. Given $n \times n$ matrices A, B, and C, where $A_{i,j}$ represents the flow between facilities i and j, $B_{i,j}$ is the distance between locations i and j, and $C_{i,j}$ is the fixed cost of assigning facility i to location j, the problem is to find the permutation $\{\pi(1), \pi(2), \ldots, \pi(n)\}$ of the index set $\{1, 2, \ldots, n\}$ that minimizes the following objective:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} A_{i,j}B_{\pi(i),\pi(j)} + \sum_{i=1}^{n} C_{i,\pi(i)}.$$

An alternative matrix-form representation is as follows:

$$\text{QAP}(A,B,C,): \quad \min \operatorname{tr}(AXB + C)X^{T}, \quad \text{s.t. } X \in \prod,$$

where $\operatorname{tr}(\cdot)$ represents the trace and $\prod$ is the set of $n \times n$ permutation matrices.

The practical difficulty of solving instances of the QAP to optimality grows rapidly with n. As recently as 1998, only the second-largest problem (n = 25) from the standard "Nugent" benchmark set [13] had been solved, and this effort required a powerful parallel platform [14]. In June 2000, a team consisting of Kurt Anstreicher and Nate Brixius (University of Iowa) and metaneos investigators Jean-Pierre Goux and Jeff Linderoth solved the largest of the Nugent problems – the n = 30 instance known as nug30 – verifying that a solution obtained earlier from a heuristic was optimal [15]. They devised a branch-and-bound algorithm based on a convex quadratic programming

relaxation of QAP, implemented it using MW, and ran it on a Condor-based computational grid spanning eight institutions. The computation used over 1,000 worker processors at its peak, and ran for a week. It was solving linear assignment problems (the core computational operation in the algorithm) at the rate of nearly a million per second during this period.

In the remainder of this section, we outline the various theoretical, heuristic, and computational ingredients that combined to make this achievement possible.

The convex quadratic programming (QP) relaxation of QAP proposed by Anstreicher and Brixius [16] yields a lower bound on the optimal objective that is tighter than alternative bounds based on projected eigenvalues or linear assignment problems. Just as important, an approximate solution to the QP relaxation can be found at reasonable cost by applying the Frank-Wolfe method for quadratic programming. Each iteration of this method requires only the solution of a dense linear assignment problem – an inexpensive operation. Hence, the Frank-Wolfe method is preferable in this context to more sophisticated quadratic programming algorithms; its slow asymptotic convergence properties are not important because only an approximate solution is required.

The QAP is solved by embedding the QP relaxation scheme in a branch-and-bound strategy. At each node of the branch-and-bound tree, some subset of the facilities is assigned to certain locations – in the nodes at level k of the tree, exactly k such assignments have been made. At a level-k node, a reduced QAP can be formulated in which the unassigned part of the permutation (which has $n - k$ components) is the unknown. The QP relaxation can then be used on this reduced QAP to find an approximate lower bound on its solution, and therefore on the cost of all possible permutations that include the k assignments already made at this node. If the bound is greater than the cost of the best permutation found to date (the incumbent), the subtree rooted at this node can be discarded. Otherwise, we need to decide whether and how to branch from this node.

Branching is performed by choosing a facility and assigning it to each location in turn (row branching) or by choosing a location and assigning each of the remaining facilities to it in turn (column branching). However, it is not always necessary to examine all possible $n - k$ children of a level-k node; some of them can be eliminated immediately by using information from the dual of the QP relaxation. In fact, one rule for deciding the next node from which to branch at level k of the tree is to choose the node that yields the fewest children. A more expensive branching rule, using a *strong branching* technique, is employed near the root of the tree (k smaller). Here, the consequence of fixing each one of a collection of promising facilities (or locations) is evaluated by provisionally making the assignment in question and solving the corresponding QP relaxation. Estimates of lower bounds are then obtained for the grandchildren of the current node, and these are summed. The node for which this summation is largest is chosen as the branching facility (location).

The branching rule and the parameters that govern the execution of the branching rule are chosen according to the level in the tree and also the *gap*, which measures the closeness of the lower bound at the current node to the incumbent objective. When the gap is large at a particular node, it is likely that exploration of the subtree rooted at this node will be a costly process. Use of a more elaborate (and expensive) branching rule tends to ensure that exploration of unprofitable parts of the subtree is avoided, thereby reducing overall run time.

Parallel implementation of the branch-and-bound technique uses an approach not unlike the FATCOP code for integer programming. Each worker is assigned the root node of a subtree to explore, in a depth-first fashion, for a given amount of time. When its time expires, it returns unexplored nodes from its subtree to the master, together with any new incumbent information. The pool of tasks on the master is ordered by the gap, so that nodes with smaller gaps (corresponding to subtrees that should be less difficult to explore) are assigned first. To reduce the number of easy tasks returned to the master, a "finish-up" heuristic permits a worker extra time to explore its subtree if its gap becomes small.

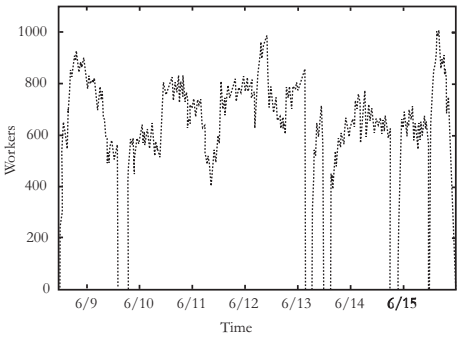
Exploitation of the symmetries that are present in many large QAPs is another important factor in making solution of nug30 and other large problems a practical proposition. Such symmetries arise when the distance matrix is derived from a rectangular grid. Symmetries can be used, for instance, to decrease the number of child nodes that need to be formed (to considerably fewer than $n - k$ children at a level-k node).

Prediction of the performance profile of a run is also important in tuning algorithmic parameters and in estimating the amount of computational resources needed to tackle the problem. An estimation procedure due to Knuth was enhanced to allow prediction of the number of nodes that need to be evaluated at each level of the branch-and-bound tree, for a specific problem and specific choices of the algorithmic parameters.

Figure 2 shows the number of workers used during the course of the nug30 run in early June 2000. As can be seen from this graph, the run was halted five times – twice because of failures in the resource management software and three times for system maintenance – and restarted each time from the latest master checkpoint.

In the weeks since the nug30 computation, the team has solved three more benchmark problems of size $n = 30$ and $n = 32$, using even larger computational grids. Several outstanding problems of size $n = 36$ derived from a backboard wiring application continue to stand as a challenge to this group and to the wider combinatorial optimization community.

Figure 2: Number of Workers during nug30 Computation



5 Stochastic Programming

The two-stage stochastic linear programming problem with recourse can be formulated as follows:

$$\min_x \; Q(x) \stackrel{\text{def}}{=} c^T x + \sum_{i=1}^{N} p_i Q_i(x)$$
$$\text{subject to} \quad Ax = b, \; x \geq 0,$$

where

$$Q_{i(x)} \stackrel{\text{def}}{=} \min_{y_i} q_i^T y_i \text{ s.t. } Wy_i = h_i - T_i x, \; y_i \geq 0.$$

The uncertainty in this formulation is modeled by the data triplets (h_i, T_i, q_i), $i = 1, 2, \ldots, N$, each of which represents a possible *scenario* for the uncertain data (h, T, q). Each p_i represents the probability that scenario i is the one that actually happens; these quantities are nonnegative and sum to 1. The quantities p_i, $i = 1, 2, \ldots, N$ are nonnegative and sum to 1; p_i is the probability that scenario i is the true one.

The two-stage problem with recourse represents a situation in which one set of decisions (represented by the first-stage variables x) must be made at the present time, while a second set of decisions (represented by y_i, $i = 1, 2, \ldots, N$) can be deferred to a later time, when the uncertainty has been resolved and the true second-stage scenario is known. The objective function represents

the *expected* cost of x, integrated over the probability distribution for the uncertain part of the model.

In many practical problems, the number of possible scenarios N either is infinite (that is, the probability distribution is continuous) or is finite but much too large to allow practical solution of the full problem. In these instances, sampling is often used to obtain an approximate problem with fewer scenarios.

Decomposition algorithms are well suited to grid platforms, because they allow the computations associated with the N second-stage scenarios to be performed independently and require only modest amounts of data movement between processors. These algorithms view Q as a piecewise linear, convex function of the variables x, whose subgradient is given by the formula

$$\partial Q(x) = c + \sum_{i=1}^{N} p_i \partial Q_i(x).$$

Evaluation of each function Q_i, $i = 1, 2, \ldots, N$ requires solution of the linear program in y_i given above. One element of the subgradient $\partial Q_i(x)$ of this function can be calculated from the dual solution of this linear program.

In the metaneos project, Linderoth and Wright [17] have implemented a decomposition algorithm based on techniques from nonsmooth optimization and including various enhancements to lessen the need for the master and workers to synchronize their efforts. In the remainder of this section, we outline in turn the trust-region algorithm ATR and its convergence properties, implementation of this algorithm on the Condor grid platform using MW, and our "asynchronous" variant.

The ATR algorithm progressively builds up a piecewise-linear model function $M(x)$ satisfying $M(x) \le Q(x)$. At the kth iteration, a candidate iterate z is obtained by solving the following subproblem:

$$\min_z M(z) \quad \text{subject to } Az = b,\ z \ge 0,$$
$$\left\| z - x^k \right\|_\infty \le \Delta,$$

where the last constraint represents a trust region with radius $\Delta > 0$. The candidate z becomes the new iterate x^{k+1} if the decrease in objective $Q(x^k) - Q(z)$ is a significant fraction of the decrease $Q(x^k) - M(z)$ promised by the model function. Otherwise, no step is taken. In either case, the trust-region radius Δ may be adjusted, function and subgradient information about Q at z is used to enhance the model M, and the subproblem is solved again. The algorithm uses a "multicut" variant in which subgradients for partial sums of $\sum_{i=1}^{N} Q_i(z)$ are included in the model separately, allowing a more accurate model to be constructed in fewer iterations.

In the MW implementation of the ATR algorithm, the function and subgradient information defining M is accumulated at the master processor, and the subproblem is solved on this processor. (Since M is always piecewise linear and the trust region is defined by an ∞-norm, the subproblem can be formulated as a linear program.) Most of the computational work in the algorithm involves solution of the N second-stage linear programs in y_i, from which we obtain $Q_i(z)$ and $\partial Q_i(z)$. This work is distributed among T tasks, to be executed in parallel, where each task requires solution of a "chunk" of N/T second-stage linear programs.

The use of chunking allows problems with very large N to make efficient use of a fairly large number of processors. However, the approach still requires evaluation of all the chunks for $Q(z)$ to be completed before deciding whether to accept or reject z as the next iterate. It is possible that one chunk will be processed much more slowly than the others-its computation may have been interrupted by the workstation's owner reclaiming the machine, for instance. All the other workers in the pool will be left idle while waiting for evaluation of this chunk to complete.

The ATR method maintains not just a single candidate for the next iterate but rather a basket ß containing 5 to 20 possible candidates. At any given time, the workers are evaluating chunks of second-stage problems associated with one or other of these basket points. ATR also maintains an "incumbent" x^I, which is the current best estimate of the solution and is a point for which $Q(x^I)$ is known. When all the chunks for one of the basket points z have been evaluated, $Q(z)$ is compared with the incumbent objective $Q(x^I)$ and with the decrease predicted by the model function M at the time z was generated. As a result, either z becomes the new incumbent and x^I is discarded, or x^I remains the incumbent and z is discarded. In either case, a vacancy is created in the basket ß. To fill the vacancy, a new candidate iterate z' is generated by solving a subproblem with the trust-region constraint centered on the incumbent, that is,

$$\left\| z' - x^I \right\|_\infty \le \Delta.$$

We show results obtained for sampled instances of problems from the stochastic programming literature, using the MW implementation of ATR running on a Condor pool. The SSN problem described in [18] arises in design of a network for private-line telecommunications services. In this model, each of 86 parameters representing demand can independently take on 3 to 7 values, giving a total of approximately $N = 10^{70}$ scenarios. Sampling is used to obtain problems with more modest values of N, which are then solved with ATR.

Table 2: SSN, with $N = 10{,}000$

Run	Iter.	Procs.	Eff.	Time(min.)
L	255	19	.46	398
ATR-1	47	19	.35	130
ATR-10	164	71	.57	43

Results for an instance of SSN with $N = 10000$ are shown in Table 2. When written out as a linear program in the unknowns $(x, y_1, y_2, \ldots, y_N)$, this problem has approximately 1,750,000 rows and 7,060,000 columns. Table 2 compares three algorithms. The first is an L-shaped method (see [19]), which obtains its iterates from a model function M but does not use a trust region or check sufficient decrease conditions. (The implementation described here is modified to improve parallelism, in that it does not wait for all the chunks for the current point to be evaluated before calculating a new iterate.) The second entry in Table 2 is for the synchronous trust-region approach (which is equivalent to ATR with a basket size of 1), and the third entry is for ATR with a basket size of 10. In all cases, the second-stage evaluations were divided into 10 chunks, and 50 partial subgradients were added to M at each iteration. The table shows the average number of processors used during the run, the proportion of time for which these processors were kept busy, and the wall clock time required to find the solution.

The trust-region approaches were considerably faster than the L-shaped approach, indicating that the need for sound algorithms remains as keen as ever in a parallel environment; we cannot rely on raw computing power to do all the work. The benefits of asynchronicity can also be seen. When ATR has a basket size of 10, it is able to use a larger number of processors and takes less time to complete, even though the number of iterates increases significantly over the synchronous trust-region approach.

The real interest lies, however, not in solving single sampled instances of SSN, but in obtaining high-quality solutions to the underlying problem (the one with 10^{70} scenarios). ATR gives a valuable tool that can be used in conjunction with variance reduction and verification techniques to yield such solutions.

Table 3: storm, with $N = 250{,}000$

Run	Iter.	Procs.	Eff.	Time(min.)
ATR-1	25	67	.57	211
ATR-5	57	86	.96	229

Finally, we show results from the "storm" problem, which arises from a cargo flight scheduling application [20]. The ATR implementation was used to solve a sampled instance with N = 250,000, for which the full linear program has approximately 132,000,000 rows and 315,000,000 columns. The results in Table 3 show that this huge linear program with nontrivial structure can be solved in less than 4 hours on a computational platform that costs essentially nothing. Because the second-stage work can be divided into a much larger number of chunks than for SSN – 125 chunks, rather than 10 – the synchronous trust-region algorithm is able to make fairly effective use of an average of 67 processors and requires less wall clock time than ATR with a basket size of 5.

Conclusions

Our experiences in the metaneos project have shown that cheap, powerful computational grids can be used to tackle large optimization problems of various types. These results have several interesting implications. In an industrial or commercial setting, the results demonstrate that one may not have to buy powerful computational servers to solve many of the large problems arising in areas such as scheduling, portfolio optimization, or logistics; the idle time on employee workstations (or, at worst, an investment in a modest cluster of PCs) may do the job. For the optimization research community, our results motivate further work on parallel, grid-enabled algorithms for solving very large problems of other types. The fact that very large problems can be solved cheaply allows researchers to better understand issues of "practical" complexity and of the role of heuristics. In stochastic optimization, higher-quality solutions can be found, and improvements to sampling methodology can be investigated.

Work remains to be done in making the grid infrastructure robust enough for general use. The logistics of assembling a grid – issues of security and shared ownership – remain challenging. The vision of a computational grid that is as easy to tap into as the electric power grid remains far off, though metaneos gives a glimpse of the way in which optimizers could exploit such a system.

We have investigated just a few of the problem classes that could benefit from solution on computational grids. Global optimization problems of different types should be examined further. Data-intensive applications (from tomography and data mining) represent a potentially huge field of work, but these require a somewhat different approach from the compute-intensive applications we have considered to date.

We hope that optimizers of all flavors, along with grid computing experts and applications specialists, will join the quest. There's plenty of work for all!

Acknowledgements

The metaneos project was funded by National Science Foundation grant CDA-9726385 and was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

For more information on the metaneos project, see <www.mcs.anl.gov/metaneos>. Information on Condor can be found at <www.cs.wisc.edu/condor> and on Globus at <www.globus.org>.

References

[1] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

[2] J. Basney, M. Livny, and T. Tannenbaum. High throughput computing with Condor. *HPCU News*, 1(2), 1997.

[3] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP*, 11(1), June 1997.

[4] I. Foster and C. Kesselman. The Globus project: A status report. In P*roceedings of the Heterogeneous Computing Workshop*, pages 4-18. IEEE Press, 1998.

[5] J.-P. Goux, J. T. Linderoth, and M. Yoder. Metacomputing and the master-worker paradigm. Preprint MCS/ANL-P792-0200, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., February 2000.

[6] J.-P. Goux, S. Kulkarni, J. T. Linderoth, and M. Yoder. An enabling framework for master-worker applications for the computational grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 43-50, Pittsburgh, Pennsylvania, August 2000.

[7] J. Eckstein. Parallel branch-and-bound methods for mixed-integer programming on the CM-5. *SIAM Journal on Optimization*, 4(4):794-814, 1994.

[8] R. E. Bixby, W. Cook, A. Cox, and E. K. Lee. Computational experience with parallel mixed integer programming in a distributed environment. *Annals of Operations Research*, 90:19-43, 1999.

[9] J. T. Linderoth. *Topics in Parallel Integer Optimization*. Ph.D. thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1998.

[10] J. Eckstein, C. A. Phillips, and W. E. Hart. PPICO: An object-oriented framework for parallel branch and bound. Research Report RRR-40-2000, RUTCOR, August 2000.

[11] Q. Chen, M. C. Ferris, and J. T. Linderoth. FATCOP 2.0: Advanced features in an opportunistic mixed integer programming solver. Technical Report Data Mining Institute Technical Report 99-11, Computer Sciences Department, University of Wisconsin at Madison, 1999. To appear in *Annals of Operations Research*.

[12] N. Robertson, D. P. Sander, P. D. Seymour, and R. Thomas. A new proof of the four color theorem. *Electronic Research Announcements of the American Mathematical Society*, 1996.

[13] 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.

[14] A. Marzetta and A. Brünger. A dynamic-programming bound for the quadratic assignment problem. In *Computing and Combinatorics: 5th Annual International Conference, COCOON '99*, volume 1627 of *Lecture Notes in Computer Science*, pages 339-348. Springer, 1999.

[15] K. Anstreicher, N. Brixius, J.-P. Goux, and J. T. Linderoth. Solving quadratic assignment problems on computational grids. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., October 2000.

[16] K. M. Anstreicher and N. Brixius. A new bound for the quadratic assignment problem based on convex quadratic programming. Technical report, Department of Management Sciences, The University of Iowa, 1999.

[17] J. T. Linderoth and S. J. Wright. Implementing decomposition algorithms for stochastic programming on a computational grid. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill, 2000. In preparation.

[18] S. Sen, R. D. Doverspike, and S. Cosares. Network planning with random demand. *Telecommunications Systems*, 3:11-30, 1994.

[19] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research. Springer, 1997.

[20] J. M. Muley and A. Ruszczyński. A new scenario decomposition method for large scale stochastic optimization. *Operations Research*, 43:477-490, 1995.

CONFERENCE Calendar

1st Cologne Twente Workshop on Graphs and Combinatorial Optimization
June 6-8, 2001, University of Cologne, Cologne, Germany.

URL: http://www.zaik.uni-koeln.de/AFS/conferences/CTW2001/CTW2001.html

IPCO VIII
June 13-15, 2001, Utrecht, The Netherlands.

URL: http://www.cs.uu.nl/events/ipco2001
email: ipco2001@cs.uu.nl

INFORMS International 2001
June 17-20, 2001, Maui, Hawaii, USA.

URL: http://www.informs.org/Conf/Hawaii2001

SIAM Annual Meeting
July 9-13, 2001, San Diego, California, USA.

URL: http://www.siam.org/meetings/an01/

Symposium on OR 2001
September 3-5, 2001, Gerhard-Mercator-Universität, Duisburg, Germany.

URL: http://www.uni-duisburg.de/or2001/

INFORMS Annual Meeting
November 4-7, 2001, Miami Beach, Florida, USA.

URL: http://www.informs.org/Conf/Miami2001

ANNOUNCEMENT

9th International Conference on Stochastic Programming

Berlin, Germany, August 25-31, 2001
(held under the auspices of the Committee on Stochastic Programming of the Mathematical Programming Society)

Contact Information. Web site: http://www.mathematik.hu-berlin.de/SP01; E-mail: sp01@mathematik.hu-berlin.de; Mailing Address: SP01, c/o Prof. W. Roemisch, Institute of Mathematics, Humboldt-University Berlin, 10099 Berlin, Germany

Support. Humboldt-Universitaet Berlin, Gerhard-Mercator-Universitaet Duisburg, Weierstrass-Institut fuer Angewandte Analysis und Stochastik (WIAS), Konrad-Zuse-Zentrum fuer Informationstechnik Berlin (ZIB), Deutsche Forschungsgemeinschaft

Scope. Stochastic programming addresses the optimization of decision making under uncertainty. Several branches of mathematics and its practical applications inspire it. Studies in stochastic programming are directed to model building, theoretical analysis of models, design of algorithms, software development, and practical implementation. The 9th International Conference on Stochastic Programming will be a forum to discuss recent advances, the state of the art, and future developments of the field.

Topics of the conference. modeling techniques, scenario generation, two-stage and chance constrained models, dynamic stochastic optimization, integer and combinatorial optimization under uncertainty, risk management, stability analysis, estimation and simulation, approximation and bounding, decomposition methods, stochastic programming models in finance, and practical applications (e.g., in energy and logistics).

International Scientific Committee. J.R. Birge (USA), M.A.H. Dempster (Great Britain), J. Dupacova (Czech Republic), P. Kall (Switzerland), A.J. King (USA), A. Prekopa (Hungary/USA), G. Pflug (Austria), W. Roemisch (Germany), A. Ruszyński (USA), R. Schultz (Germany), S. Sen (USA), A. Shapiro (USA), S.W. Wallace (Norway), R. J-B Wets (USA), W.T. Ziemba (Canada)

Invited Plenary Speakers. Hans Foellmer (Berlin), Suvrajeet Sen (Tucson), Alexander Shapiro (Atlanta), Stein W. Wallace (Trondheim), Roger J-B Wets (Davis)

Organizing Committee. K. Frauendorfer (St. Gallen), R. Henrion (WIAS Berlin), P. Kall (Zuerich), K. Marti (Muenchen), G. Pflug (Wien), W. Roemisch (HU Berlin), R. Schultz (Duisburg), M.C. Steinbach (ZIB Berlin), S. Vogel (Ilmenau)

Local Organizers. W. Roemisch, R. Schultz, N. Groewe-Kuska, R. Henrion, J. Kerger, H. Lange, A. Moeller, I. Nowak, M.C. Steinbach, I. Wegner

Location. Humboldt-University Berlin (Main Building), Unter den Linden 6, 10117 Berlin

Schedule. Tutorials: August 25-26, 2001; International Conference: August 27-31, 2001; Opening Session: August 27, 2001, 9:30 am

Fees. Conference fee: 400.- DM (before May 31, 2001), 450.- DM; 150.- DM (student); Tutorial fee: 100.- DM

Deadlines. February 15, 2001 Starting Online Registration; April 15, 2001 Abstract Submission; April 30, 2001 Notification of Acceptance; May 31, 2001 Early Registration

EURO 2001: The XVIII-th Euro Conference on Operations Research

Why Visit Euro 2001: Meeting OR friends in convenient and well-equipped conference center; participating in and contributing to high quality theory and practice sessions; learning more about smart logistics and innovative operations during special conference sessions, company visits, and excursions to the largest port in the world; reduced fee to a post conference seminar on Quantitative Financial Risk Management in Amsterdam; enjoying social events in Europe's cultural capital of 2001
Where: Erasmus University Rotterdam, The Netherlands
When: July 9-11, 2001
How to Register: The most convenient way to register is via our web site (www.euro2001.org). Alternatively, you may order a registration card via our e-mail address (info@euro2001.org).
Fees: Early: Euro 300 for Non-Students, Euro 200 for Students; Late: Euro 350 for Non-Students, Euro 250 for Students.
Abstract Submission: Please refer to our web site for submission instructions, or ask for instructions on paper via our e-mail address (info@euro2001.org).
Further Information: E-mail: info@euro2001.org; web site: www.euro2001.org
Deadlines: The early registration deadline is May 1, 2001; the abstract submission deadline is March 1, 2001. Please note that submissions that have been received after the deadline cannot be accepted.

Abstracts can be submitted via the web site (www.euro2001.org), which also contains a list of topics and submission instructions.
We hope to welcome you at EURO 2001!
www.euro2001.org

Ettore Majorana Centre for Scientific Culture International School of Mathematics G. Stampacchia Workshop

High Performance Algorithms and Software for Nonlinear Optimization

Erice, Italy
June 30 - July 8, 2001

Objectives: In the first year of a new century, the Workshop aims to assess the past and to discuss the future of Nonlinear Optimization. Recent achievements and promising research trends will be highlighted. Algorithmic and high performance software aspects will be emphasized, along with theoretical advances and new computational experiences. Contributions from different and complementary standpoints, covering several fields of numerical optimization and its applications are expected.
Topics: Topics include, but are not limited to: constrained and unconstrained optimization, global optimization, derivative-free methods, interior point techniques for nonlinear programming, large scale optimization, linear and nonlinear complementarity problems, nonsmooth optimization, neural networks and optimization, applications of nonlinear optimization.
Lectures: The Workshop will consist of invited lectures (1 hour) and contributed lectures (1/2 hour). Invited lecturers who have confirmed the participation are: P. Boggs, A. R. Conn, J. Dennis, Y. G. Evtushenko, F. Facchinei, R. Fletcher, J. C. Gilbert, P. E. Gill, D. Goldfarb, S. Lucidi, J. J. Moré, S. G. Nash, E. J. Polak, E. Sachs, D. Shanno, Ph. L. Toint, V. J. Torczon, J. P. Vial, R. J. Vanderbei, M. H. Wright, S. Wright, J. Z. Zhang.
Proceedings: Edited proceedings including the invited lectures and a selection of contributed lectures will be published.

Further information can be found online (http://www.dis.uniroma1.it/~or/erice) or requested via e-mail (erice@dis.uniroma1.it).

Franco Giannessi, School Director
Gianni Di Pillo and **Almerico Murli**, Workshop Directors

International Congress of Mathematicians 2002

Beijing, China
August 20-28, 2002

The next International Congress of Mathematicians will take place in Beijing, People's Republic of China, August 20-28, 2002. For pre-registration forms and additional information, please visit the web site (http://www.icm2002.org.cn), or e-mail the organizers (icmsec@beijing.icm2002.org.cn). Pre-registrants will automatically receive progress reports on the upcoming Congress as well as final registration materials.

The International Congress of Mathematicians takes place every four years and is supported by the International Mathematical Union (IMU). For information regarding the IMU, please visit its web site (http://mathunion.org/).

Call for Papers

MATHEMATICAL PROGRAMMING Series B

Issue on Algebraic and Geometric Methods in Discrete Optimization

We invite research articles on algebraic and geometric methods in discrete optimization and related areas for a forthcoming issue of Mathematical Programming, Series B. The goal is to provide a common forum for non-traditional methods in the theory and practice of discrete optimization. Of special interest are methods from the geometry of numbers, topology, commutative algebra, algebraic geometry, probability theory, computer science and combinatorics.

Deadline for submission of full papers: April 30, 2001.

We aim at completing a first review of all papers by September 30, 2001.

All submissions will be refereed according to the usual standards of Mathematical Programming. Information about this issue can be obtained from the guest editors for this volume (or at www.math.washington.edu/~thomas).

Karen Aardal, Department of Computer Science, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands; e–mail: aardal@cs.uu.nl **and Rekha R. Thomas, Department of Mathematics, Box 354350, University of Washington, Seattle, WA 98195-4350, U.S.A.; e-mail:** thomas@math.washington.edu

mind sharpener

We invite OPTIMA readers to submit solutions to the problems to Robert Bosch (bobb@cs.oberlin.edu). The most attractive solutions will be presented in a forthcoming issue.

Painting by Numbers

Robert A. Bosch

September 28, 2000

A *paint-by-numbers* puzzle consists of an $m \times n$ grid of pixels (the canvas) together with $m + n$ cluster-size sequences, one for each row and column. The goal is to paint the canvas with a picture that satisfies the following constraints:

1. Each pixel must be black or white.
2. If a row or column has cluster-size sequence $s_1, s_2, \ldots s_k$, then it must contain k clusters of black pixels – the first with s_1 black pixels, the second with s_2 black pixels, and so on.

It should be noted that "first" means "leftmost" for rows and "topmost" for columns, and that rows and columns need not begin or end with black pixels.

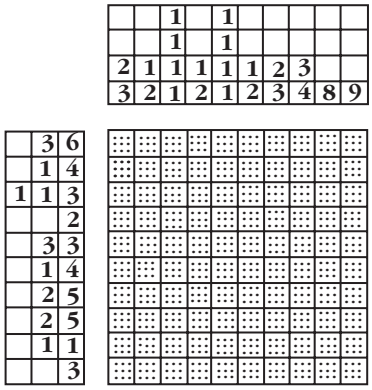


Figure 1

Small paint-by-numbers puzzles are easy to solve. One useful strategy is to alternate between analyzing the row cluster-size sequences and the column cluster-size sequences. For the puzzle displayed in Figure 1, a first pass through the rows enables us to paint 15 pixels black and one pixel white, as displayed in Figure 2. (Since row 1's cluster-size sequence is 3, 6, its first three pixels must be black, its fourth pixel must be white, and its final six pixels must be black. Since rows 7 and 8 have cluster-size sequence 2, 5, their second clusters must occupy pixels 4 through 8, 5 through 9, or 6 through 10; in each case they occupy pixels 6, 7, and 8.) A subsequent pass through the columns enables us to paint 19 more pixels black and 19 more pixels white, as displayed in Figure 3.

Four more passes through the rows and columns enable us to paint the remaining 46 pixels. The unique solution, a picture of a spaceman, is displayed in Figure 4. It should be noted that only well designed paint-by-numbers puzzles have unique solutions.

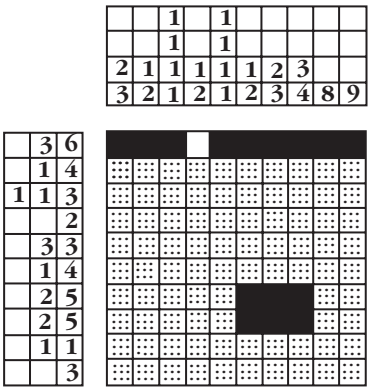


Figure 2

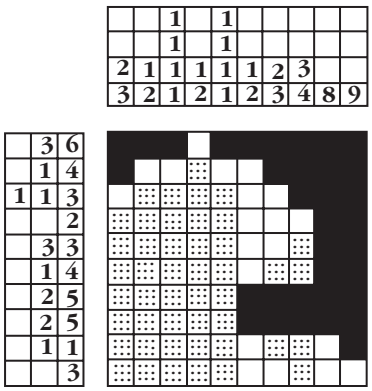


Figure 3

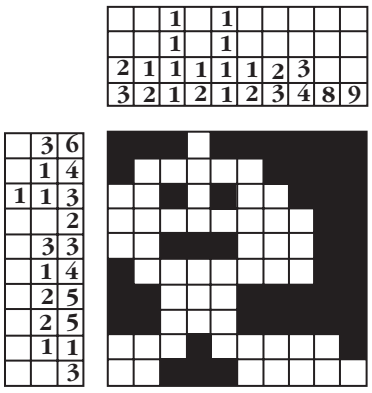


Figure 4

An IP Formulation

In this section, we describe how integer programming can be used to solve paint-by-numbers puzzles. Our approach is to think of a paint-by-numbers puzzle as a problem comprised of two interlocking tiling problems: one involving the placement of the row clusters on the canvas, and the other involving the placement of the column clusters. Note that if a pixel is painted black, it must be covered by both a row cluster and a column cluster; if it is painted white, it must be left uncovered by the row clusters and column clusters.

Notation

Let

m = the number of rows,
n = the number of columns,
k_i^r = the number of clusters in row i,
k_j^c = the number of clusters in column j,
$s_{i,1}^r, s_{i,2}^r, \ldots, s_{i,k_i^r}^r$ = the cluster-size sequence for row i,
$s_{j,1}^c, s_{j,2}^c, \ldots, s_{j,k_j^c}^c$ = the cluster-size sequence for column j,

In addition, let

$e_{i,t}^r$ = the smallest value of j such that row i's t^{th} cluster can be placed in row i with its leftmost pixel occupying pixel j,
$l_{i,t}^r$ = the largest value of j such that row i's t^{th} cluster can be placed in row i with its leftmost pixel occupying pixel j
$e_{j,t}^c$ = the smallest value of i such that column j's t^{th} cluster can be placed in column j with its topmost pixel occupying pixel i,
$l_{j,t}^c$ = the largest value of i such that column j's t^{th} cluster can be placed in column j with its topmost pixel occupying pixel i.

(The letters "e" and "l" stand for "earliest" and "latest.") In our example puzzle, row 7's first cluster must be placed so that its leftmost pixel occupies pixel 1, 2, or 3; row 7's second cluster must be placed so that its leftmost pixel occupies pixel 4, 5, or 6. In other words, $e_{7,1}^r = 1$, $l_{7,1}^r = 3$, $e_{7,2}^r = 4$ and $l_{7,2}^r = 6$.

Variables

Our formulation uses three sets of variables. One set specifies which pixels are painted black. For each $1 \le i \le m$ and $1 \le j \le n$, let

$$z_{i,j} = \begin{cases} 1 & \text{if row } i\text{'s } j^{\text{th}} \text{ pixel is painted black,} \\ 0 & \text{if row } i\text{'s } j^{\text{th}} \text{ pixel is painted white} \end{cases}$$

The other two sets of variables are concerned with the placements of the row and column clusters. For each $1 \le i \le m$, $1 \le t \le k_i^r$, and $e_{i,t}^r \le j \le l_{i,t}^r$, let

$$y_{i,t,j} = \begin{cases} 1 & \text{if row } i\text{'s } t^{\text{th}} \text{ cluster is placed in row } i \text{ with its leftmost pixel occupying pixel } j, \\ 0 & \text{if not.} \end{cases}$$

For each $1 \le j \le n$, $1 \le t \le k_j^c$, and $e_{j,t}^c \le i \le l_{j,t}^c$, let

$$x_{j,t,i} = \begin{cases} 1 & \text{if column } j\text{'s } t^{\text{th}} \text{ cluster is placed in column } j \text{ with its topmost pixel occupying pixel } i, \\ 0 & \text{if not.} \end{cases}$$

Cluster Constraints

To ensure that row i's t^{th} cluster appears in row i exactly once, we impose

$$\sum_{j=e_{i,t}^r}^{l_{i,t}^r} y_{i,t,j} = 1.$$

The sum is over all pixels j such that row i's t^{th} cluster can be placed in row i with its leftmost pixel occupying pixel j. For the two clusters of row 7 of our example puzzle, we impose

$$y_{7,1,1} + y_{7,1,2} + y_{7,1,3} = 1$$

and

$$y_{7,2,4} + y_{7,2,5} + y_{7,2,6} = 1.$$

To guarantee that row i's $(t+1)^{\text{th}}$ cluster is placed to the right of its t^{th} cluster, we impose

$$y_{i,t,j} \le \sum_{j'=j+s_{i,t}^r+1}^{l_{i,t+1}^r} y_{i,t+1,j'}$$

$$\text{for each } e_{i,t}^r + 1 \le j \le l_{i,t}^r$$

For row 7 of our example puzzle, we impose

$$y_{7,1,2} \le y_{7,2,5} + y_{7,2,6}$$

and

$$y_{7,1,3} \le y_{7,2,6}.$$

Similar constraints (involving the $x_{j,t,i}$'s) are needed for the column clusters.

Double Coverage Constraints

To guarantee that each black pixel is covered by both a row cluster and a column cluster, we impose, for each $1 \le i \le m$ and $1 \le j \le n$, the following pair of inequalities:

$$z_{i,j} \le \sum_{t=1}^{k_i^r} \sum_{j'=\min\{e_{i,t}^r,\, j-s_{i,t}^r+1\}}^{\max\{l_{i,t}^r,\, j\}} y_{i,t,j'},$$

$$z_{i,j} \le \sum_{t=1}^{k_j^c} \sum_{i'=\min\{e_{j,t}^c,\, i-s_{j,t}^c+1\}}^{\max\{l_{j,t}^c,\, i\}} x_{j,t,i'}$$

The first inequality states that if row i's j^{th} pixel is painted black, then at least one of row i's clusters must be placed in such a way that it covers row i's j^{th} pixel. (The upper and lower limits of the second summation make sure that j' satisfies the inequalities $e_{i,t}^r \le j' \le l_{i,t}^r$ and $j - s_{i,t}^r + 1 \le j' \le j$. The first two hold if and only if row i's t^{th} cluster can be placed in row i with its leftmost pixel occupying pixel j'. The last two hold if and only if row i's j^{th} pixel is covered when row i's t^{th} cluster is placed in row i with its leftmost pixel occupying j'.) The other one makes sure that if row i's j^{th} pixel is painted black, then at least one of column j's clusters covers it. In our example problem, for row 7's fifth pixel, we impose

$$z_{7,5} \le y_{7,2,4} + y_{7,2,5}$$

and

$$z_{7,5} \le x_{5,3,7} + x_{5,4,7}.$$

Finally, we include constraints that prevent white pixels from being covered by the row clusters or column clusters. For each $1 \le i \le m$, each $1 \le j \le n$, each $1 \le t \le k_i^r$ and each j' that satisfies the inequalities $j - s_{i,t}^r + 1 \le j' \le j$ and $e_{i,t}^r \le j' \le l_{i,t}^r$, we impose

$$z_{i,j} \ge y_{i,t,j'}.$$

And for each $1 \le i \le m$, $1 \le j \le n$, $1 \le t \le k_j^c$, and each i' that satisfies the inequalities $e_{j,t}^c \le i' \le l_{j,t}^c$ and $i - s_{j,t}^c + 1 \le i' \le i$, we impose

$$z_{i,j} \ge x_{j,t,i'}.$$

In our example problem, for row 7's fifth pixel, we impose

$$z_{7,5} \ge y_{7,2,4},$$
$$z_{7,5} \ge y_{7,2,5},$$
$$z_{7,5} \ge x_{5,3,7},$$
$$z_{7,5} \ge x_{5,4,7}.$$

Objective Function

There is no need for an objective function here.

Problems

Interested readers may enjoy trying to solve the following problems:

1. Try to improve the IP formulation. To obtain our code for constructing Cplex ".lp" files for the IP formulation, point your browser to www.oberlin.edu/~math/faculty/people/bosch.html

   and scroll down to the *Optima Mindsharpener* section.

2. Is there a better method for solving paint-by-numbers puzzles?

3. Solve the paint-by-numbers puzzle displayed in Figure 5. This puzzle was designed by Won Yoon Jo. Its solution is quite lovely, and it is perhaps the most difficult 20 x 20 puzzle on the Paint-by-Numbers Web Page (much more difficult than many of the 30 x 30 and 40 x 40 puzzles). To reach the Paint-by-Numbers Web Page, point your browser to www02.so-net.ne.jp/~kajitani/cgi-bin/pbn.cgi/index.html

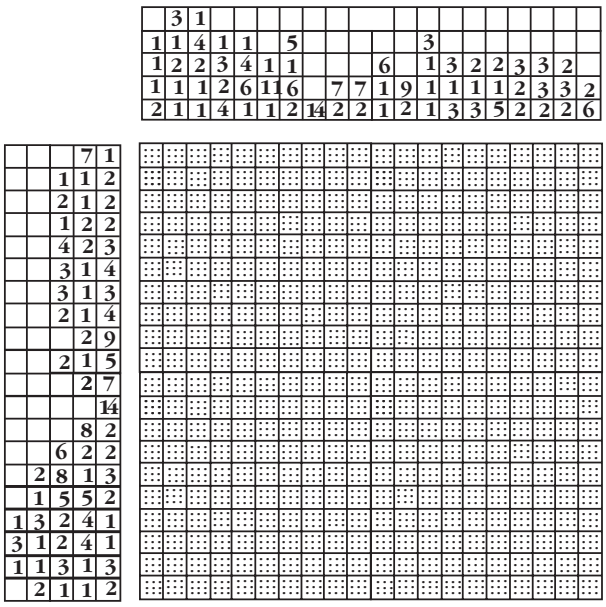


Figure 5

Gallimaufry

The deadline for the next issue of OPTIMA will be June 30, 2001. Contributions should be sent to the new editor, Jens Clausen (jc@imm.dtu.dk).

This is the last issue by the current editorial team! We wish to thank all of the authors that have contributed with articles during the past years.

We also wish to thank Elsa Drake for her fantastic job as the designer of OPTIMA. She has been the designer since 1985, helping make it one of the most important and recognized newsletters in the mathematical sciences community. The quality emphasis of the society has been captured in her creative graphics and layout. The MPS expresses its appreciation for her excellent work and wishes her well. A warm thank you for all of their assistance is also directed to Don Hearn, the founding editor, and Patsy Messinger.

Finally, we wish the incoming Editor Jens Clausen and his team GOOD LUCK!

Application for Membership

I wish to enroll as a member of the Society.

My subscription is for my personal use and not for the benefit of any library or institution.

☐ I will pay my membership dues on receipt of your invoice.

☐ I wish to pay by credit card (Master/Euro or Visa).

CREDIT CARD NO. EXPIRATION DATE

FAMILY NAME

MAILING ADDRESS

TELEPHONE NO. TELEFAX NO.

EMAIL

SIGNATURE ▶

Mail to:

Mathematical Programming Society
3600 University City Sciences Center
Philadelphia PA 19104-2688 USA

Cheques or money orders should be made payable to The Mathematical Programming Society, Inc. Dues for 2001, including subscription to the journal ***Mathematical Programming***, are US $75.
Student applications: Dues are one-half the above rate. Have a faculty member verify your student status and send application with dues to above address.

Faculty verifying status

Institution

Springer ad (insert film)

O P T I M A

MATHEMATICAL PROGRAMMING SOCIETY

UNIVERSITY OF
FLORIDA

Center for Applied Optimization
371 Weil
PO Box 116595
Gainesville FL 32611-6595 USA

FIRST CLASS MAIL