N°58

OPTIMA

Mathematical Programming Society Newsletter

JUNE 1998

David Karger was awarded the 1997 Tucker Prize by the Society for his thesis *Random Sampling in Graph Optimization Problems*. The jury found his contribution, on the interface between computer science and optimization, deep and elegant. To make David's work even better known among the membership, I invited him to write a feature article on some of his very interesting techniques and results.
David Karger obtained an A.B. degree from Harvard University, a Math Certificate from Cambridge University, and a Ph.D. degree from Stanford University. Apart from the Tucker Prize, David has been awarded numerous other fellowships and prizes, such as the ACM Doctoral Dissertation award, an Alfred P. Sloane Foundation Fellowship, and a David and Lucille Packard Foundation Fellowship. His main research interests are analysis of algorithms and information retrieval, on which he has written numerous papers that appeared in both conference proceedings and journals. Since 1995, David has been assistant professor at the Laboratory for Computer Science at MIT.
If you want to find out more about David's research activities, his home page (see note, below) is an excellent source!
– Karen Aardal

Randomization in Graph Optimization Problems: A Survey

*David Karger**

Abstract

Randomization has become a pervasive technique in combinatorial optimization. We survey our thesis and subsequent work, which uses four common randomization techniques to attack numerous optimization problems on undirected graphs.

1 Introduction

Randomization has become a pervasive technique in combinatorial optimization. Randomization has been used to develop algorithms that are faster, simpler, and/or better-performing than previous deterministic algorithms. This article surveys our thesis [Kar94], which presents randomized algorithms for numerous problems on undirected graphs. Our work uses four important randomization techniques: Random Selection, which lets us easily choose a "typical" element of a set, avoiding rare "bad" elements; Random Sampling, which provides a quick way to build a small, representative subproblem of a larger problem for quick analysis; Randomized Rounding, which lets us transform fractional problem solutions into integral ones; and Monte Carlo Simulation, which lets us estimate the probabilities of interesting events.

We apply these techniques to numerous optimization problems on undirected graphs. The graph is one of the most common structures in computer science and optimization, modeling among other things roads, communication and transportation networks, electrical circuits, relationships between individuals, hypertext collections, resource allocations, project plans, database and program dependencies, and parallel architectures. Among the graph problems we address are finding a minimum spanning tree, finding a maximum flow, determining the connectivity (minimum cut) of a graph, network design, graph coloring, and estimating the reliability (disconnection probability) of a network with random edge failures. A great deal of work has been done on all of these problems. Due to space limitations, we are unable to discuss all of this related work. Such a discussion can be found in our thesis.

It is extremely important to note that we are not carrying out "expected case" analysis of algorithms running on random inputs. Rather, we consider *worst case* inputs and use random choices by the algorithm to solve them efficiently.

We begin this article with a survey of many of our applications of the four randomization techniques, sketching the methods, the problems, and the resulting algorithms. Afterwards, we will present a more detailed discussion of a few algorithms and proofs which will hopefully give some flavor of our work.

PAGE TWO ▶

*MIT Laboratory for Computer Science, Cambridge, MA 02138
e-mail: karger@lcs.mit.edu
URL: http://theory.lcs.mit.edu/~karger
Work supported in part by ARPA contract N00014-95-1-1246 and NSF contract CCR-9624239, and Fellowships from the Alfred P. Sloane and David and Lucille Packard Foundations.

1.1 Notation

We address only undirected graphs; directed graphs have so far not succumbed to the techniques we apply here. Throughout our discussion, we will consider a graph G with m edges and n vertices. Each graph edge may have a *weight* (reflecting cost or capacity) associated with it. To simplify our presentation here, we often focus on unweighted graphs (that is, graphs with all edge weights equal to one), though many of our algorithms apply equally well to weighted graphs. Unless it has parallel edges (multiple edges with the same endpoints) a graph has $m \leq \binom{n}{2}$.

The notation $\tilde{O}(f)$ denotes $O(f \log^d n)$ for some constant d.

1.2 Overview of Results

We show how randomization can be used in several ways on problems of varying degrees of difficulty. For the "easy to solve" minimum spanning tree problem, where a long line of research has resulted in ever closer to linear-time algorithms, random sampling gives the final small increment to a truly linear time algorithm. For harder problems it improves running times by a more significant factor. For example, we improve the time needed to find minimum cuts from $\tilde{O}(mn)$ to $\tilde{O}(m)$, and give the first efficient parallel algorithm for the problem. Addressing some hard *NP*-complete problems such as network design and graph coloring, where finding an exact solution in polynomial time is thought to be hopeless, we use randomized rounding to give better approximation algorithms than were previously known. Finally, for the problem of determining the reliability of a network (a #P-complete problem which is thought to be "even harder" than *NP*-complete ones) we use Monte Carlo simulation to give the first efficient approximation algorithm.

1.3 Randomized Algorithms

Our work deals with randomized algorithms. Our typical model is that the algorithm has a source of "random bits"–variables that are mutually independent and take on values 0 or 1 with probability 1/2 each. Extracting one random bit from the source is assumed to take constant time. If our algorithms use more complex operations, such as flipping biased coins or generating samples from more complex distributions, we take into account the time needed to simulate these operations in our unbiased-bit model. Event probabilities are taken over the sample space of random bit strings produced by the random bit generator. An event occurs *with high probability (w.h.p.)* if on problems of size n it occurs with probability greater than $(1 - \frac{1}{n^k})$ for some constant $k>1$, and with *low probability* if its complement occurs with high probability.

The random choices that an algorithm makes can affect both its running time and its correctness. An algorithm that has a fixed (deterministic) running time but has a low probability of giving an incorrect answer is called *Monte Carlo (MC)*. If the running time of the algorithm is a random variable but the correct answer is given with certainty, then the algorithm is said to be *Las Vegas (LV)*. Depending on the circumstances, one type of algorithm may be better than the other. However, a Las Vegas algorithm is "stronger" in the following sense.

A Las Vegas algorithm can be made Monte Carlo by having it terminate with an arbitrary wrong answer if it exceeds the time bound $f(n)$. Since the Las Vegas algorithm is unlikely to exceed its time bound, the converted algorithm is unlikely to give the wrong answer. On the other hand, there is no universal method for making a Monte Carlo algorithm into a Las Vegas one, and indeed some of the algorithms we present are Monte Carlo with no Las Vegas version apparent. The fundamental problem is that sometimes it is impossible to check whether an algorithm has given a correct answer. However, the failure probability of a Monte Carlo optimization algorithm can be made arbitrarily small by repeating it several times and taking the best answer; we shall see several examples of this below. In particular, we can reduce the failure probability so far that other unavoidable events (such as a power failure) are more likely than an incorrect answer.

2 A Survey of Techniques and Results

In this section, we provide a high level overview of the four randomization techniques and the various algorithms we have been able to develop by using them.

2.1 Random Selection

The first and simplest randomization technique we discuss is random selection. The intuition behind this idea is that a single randomly selected individual is probably a "typical" representative of the entire population. Thus, random selection provides a good way to avoid choosing rare "bad" elements. This is the idea behind Quicksort [Hoa62], where the assumption is that the randomly selected pivot will be neither ex-

Mathematical Programming has a new publisher.

We are happy to announce that starting in 1999, *Mathematical Programming* will be published by Springer-Verlag. This decision was the result of an extensive search and negotiation carried out by a committee made up of Stephen Wright, Jan Karel Lenstra and myself. The deliberations of the committee were reviewed periodically by the Mathematical Programming Society Council, with the final decision subsequently approved by the Council. The three principal reasons for the decision: Springer's willingness to accept a pricing scheme in line with society expectations; the worldwide editorial and marketing capabilities of Springer; and a modern, successful facility for providing electronic access to the journal.

The contractual details were settled just before Christmas 1997. As before, each member of the society will receive a free copy of *Mathematical Programming*. Additional benefits include:

- Each quarter, Springer-Verlag will offer a list of 50 books selected to be of particular interest to MPS. These books will be available at a discount of 25%. The list will be published in "OPTIMA" and as part of Springer's marketing information on the Internet (http://www.springer.de).
- Members of the society will have free access to the electronic version of *Mathematical Programming* via Springer's LINK service.

Springer is one of the world's best known publishers in mathematics, computer science and operations research and will launch a major international promotional campaign for our journal. Increased subscriptions generated by Springer will also increase visibility of our society and journal in related fields of research. Members of the society are urged to support this effort by encouraging "their" libraries to subscribe to the journal.

From August onwards, society members will have the chance for a "sneak preview" for papers already accepted for publication in Volume 84 for 1999. To see how the electronic journal facility is working, please look at http://link.springer.de/.

If you have any questions concerning the change, the editors to contact at Springer-Verlag are: Dr. Martin Peters, Senior Mathematics Editor, Springer-Verlag, Tiergartenstrasse 17, D-69121 Heidelberg, Germany (Peters@Springer.de); Achi Dosanjh, Mathematics Editor, Springer-Verlag, 175 Fifth Avenue, New York, NY 10010, USA (Adosanjh@Springer-ny.com); Maria Taylor, Executive Editor, Physical Sciences Journals, Springer-Verlag, 175 Fifth Avenue, New York, NY 10010, USA (Mtaylor@Springer-ny.com).

–BOB BIXBY, CHAIR OF THE PUBLICATIONS COMMITTEE

tremely large nor extremely small, and will therefore serve to separate the remaining elements into two roughly equal sized groups.

We apply this idea in a new algorithm for finding minimum cuts in undirected graphs. A *cut* is a partition of the graph vertices into two groups; the *value* of the cut is the number (or total weight) of edges with one endpoint in each group. The minimum cut problem is to identify a cut of minimum value. We distinguish the (global) minimum cut from an *s*-*t minimum cut* which is required to separate two specific vertices *s* and *t*. Finding minimum cuts is of great importance in analyzing network reliability and also plays a role in solving traveling salesman and network design problems.

We present the Recursive Contraction Algorithm (joint work with Clifford Stein [KS96]). The idea behind our algorithm is simple: a randomly selected edge is unlikely to cross a particular minimum cut, so its endpoints are probably on the same side. If we merge two vertices on the same side of this minimum cut, then we will not affect the minimum cut but will reduce the number of graph vertices by one. Therefore, we can find the minimum cut by repeatedly selecting a random edge and merging its endpoints until only two vertices remain and the minimum cut becomes obvious.

An efficient implementation of the above idea leads to a strongly polynomial $\tilde{O}(n^2)$-time algorithm for the minimum cut problem on weighted undirected graphs. In contrast, the best deterministic bound, due to Hao and Orlin [HO94], is $\tilde{O}(mn)$. Our algorithm actually finds *all* minimum cuts with high probability. It extends to enumerating approximately minimum cuts and minimum *k*-way cuts for constant *k*, as well as to constructing the *cactus* of a graph (a compact representation of all its minimum cuts). The algorithm is the first with a theoretically good parallelization. A *derandomization* of the algorithm (joint with Rajeev Motwani [KM97]) gave the first proof that there was a fast deterministic parallel algorithm for the minimum cut problem. An implementation experiment shows that the algorithm has reasonable time bounds in practice [CGK+97].

The Contraction Algorithm is Monte Carlo: it gives the correct answer with high probability, but does have a small chance of being wrong. As we are unaware of any algorithm for *verifying* that a cut is minimum, we have been unable to devise a Las Vegas version of the algorithm. This is a case where a willingness to occasionally be wrong seems to provide a significant speedup.

The Contraction Algorithm also gives an important new bound on the number of small cuts a graph may contain; this has important applications in network reliability analysis and graph sampling (see below).

In subsequent work [BK98], we used the Contraction Algorithm to efficiently solve the *graph augmentation problem*: adding the minimum possible capacity to a graph so as to increase its minimum cut to a given value (this work is joint with Andras Benczùr).

2.2 Random Sampling

A more general use of randomization than random selection is to generate small representative subproblems. The representative random sample is a central concept of statistics. It is often possible to gather a great deal of information about a large population by examining a small sample randomly drawn from it. This approach has obvious advantages in reducing the investigator's work, both in gathering and in analyzing the data.

Given an optimization problem, it may be possible to generate a small representative subproblem by random sampling (perhaps the most natural sample from a graph is a random subset of its edges). Intuitively, such a subproblem should form a microcosm of the larger problem. Our goal is to examine the subproblem and use it to glean information about the original problem. Since the subproblem is small, we can spend proportionally more time examining it than we would spend examining the original problem. In one approach that we use frequently, an optimal solution to the subproblem may be a nearly optimal solution to the problem as a whole. In some situations, such an approximation might be sufficient. In other situations, it may be easy to refine this good solution into a truly optimal solution.

Floyd and Rivest [FR75] use this approach in a fast and elegant algorithm for finding the median of an ordered set. They select a small random sample of elements from the set and show how inspecting this sample gives a very accurate estimate of the value of the median. It is then easy to find the actual median by examining only those elements close to the estimate. This algorithm, which is very simple to implement, uses fewer comparisons than any other known median-finding algorithm.

The Floyd-Rivest algorithm typifies three components needed in a random-sampling algorithm. The first is a definition of a *randomly sampled subproblem*. The second is an *approximation theorem* that proves that a solution to the subproblem is an approximate solution to the original problem. These two components by themselves will typically yield an obvious approximation algorithm with a speed-accuracy tradeoff. The third component is a *refinement algorithm* that takes the approximate solution and turns it into an actual solution. Combining these three components can yield an algorithm whose running time will be determined by that of the refinement algorithm; intuitively, refinement should be easier than computing a solution from scratch.

In an application of this approach, we present the first (randomized) linear-time algorithm for finding minimum spanning trees in the comparison-based model of computation. This result reflects joint work with Philip Klein and Robert E. Tarjan [KKT95]. A long stream of results reduced the best known deterministic time bound to almost linear [GGST86], but a linear time bound remained elusive. Our fundamental insight is that if we construct a subgraph of a graph by taking a random sample of the graph's edges, then the minimum spanning tree in the subgraph is a "nearly" minimum spanning tree of the entire graph. More precisely, very few graph edges can be used to improve the sample's minimum spanning tree. By examining these few edges, we can refine our approximation into the actual minimum spanning tree at little additional cost.

We also apply sampling to the minimum cut problem and several other related problems involving cuts in graphs, including maximum flows. The maximum flow problem is perhaps the most widely studied of all graph optimization problems, having hundreds of applications. Given vertices *s* and *t* and capacitated edges, the goal is to ship the maximum quantity of material from *s* to *t* without exceeding the capacities of the edges. The value of a graph's maximum flow is completely determined by the value of the minimum *s*-*t* cut in the graph.

We prove a cut sampling theorem that says that when we choose half a graph's edges at random we approximately halve the value of every cut. In particular, we halve the graph's connectivity and the value of all *s*-*t* minimum cuts and maximum flows. This theorem gives a random-sampling scheme for approximating minimum cuts and maximum flows: compute the minimum cut and maximum flow in a random sample of the graph edges. Since the sample has fewer edges, the computation is faster. At the same time, our sampling theorems show that this approach gives accurate estimates of the correct values. If we want to get exact solutions rather than approximations, we still can use our samples as starting points to which we can apply inexpensive refinement algorithms. This leads to a simple randomized divide-and-conquer algorithm for finding exact maximum flows.

We put our results on graph sampling into a larger framework by examining sampling from *matroids* [Kar93]. We generalize our minimum spanning tree algorithm to the problem of finding a minimum cost matroid basis, and extend our cut-sampling and maximum flow results to the problem of matroid basis packing. Our techniques actually give a paradigm that can be applied to any *packing problem* where the goal, given a collection of *feasible* subsets of a universe, is to find a maximum collection of disjoint feasible subsets. For example, in the maximum flow problem, we are attempting to send units of flow from *s* to *t*. Each such unit of flow travels along a path from *s* to *t*, so the feasible edge-sets are the *s*-*t* paths. We apply the sampling paradigm to the problem of packing disjoint bases in a matroid, and get faster algorithms for approximating and exactly finding optimum basis packings.

We have continued to apply random sampling technique in work following our thesis. Our recent results include:

►

- An $O(m\log^3 n)$-time Monte Carlo algorithm for finding minimum cuts [Kar96],
- A *compression algorithm* that lets us transform any undirected graph into a graph with $\tilde{O}(n)$ edges but roughly the same cut values, speeding up any algorithm that depends only on cut values [BK96],
- As an application, an $O(n^2\log n)$-time Monte Carlo algorithm for finding any constant factor approximation to an s-t minimum cut [BK96],
- An $\tilde{O}(m\ n)$-time Las Vegas algorithm for finding any constant factor approximation to an s-t maxflow [Kar98a],
- An $\tilde{O}(n^{2.22})$-time algorithm for finding a maximum flow in a simple (uncapacitated) graph [KL98] (joint work with Matt Levine).

All of these bounds are significantly better than the best general time bound for finding maximum flows in directed graphs ($\tilde{O}(mn)$ for a strongly polynomial bound [GT88], and recently $\tilde{O}(m^{3/2}\log U)$ for a scaling algorithm of Goldberg and Rao [GR97]). This suggests that perhaps a better bound for maximum flow can be achieved, at least on undirected graphs.

2.3 Randomized Rounding

Yet another powerful randomization technique is *randomized rounding*. This approach is used to find approximate solutions to *NP*-hard *integer programs*. These problems typically ask for an assignment of 0/1 values to variables x_i such that linear constraints of the form $\sum a_i x_i = c$ are satisfied. If we *relax* the integer program, allowing each x_i to take any real value between 0 and 1, we get a linear program that can be solved in polynomial time, giving values p_i such that $\sum a_i p_i = c$. Raghavan and Thompson [RT87] observed that we could treat the resulting values p_i as probabilities. If we randomly set x_i=1 with probability p_i and 0 otherwise, then the *expected value* of $\sum a_i x_i$ is $\sum a^i p_i = c$. Raghavan and Thompson presented techniques for ensuring that the randomly chosen values do in fact yield a sum near the expectation, thus giving approximately correct solutions to the integer program. We can view randomized rounding as a way of sampling randomly from a large space of *answers*, rather than subproblems as before. Linear programming relaxation is used to construct an answer-space in which most of the answers are good ones.

We use our graph sampling theorems to apply randomized rounding to *network design problems*. Such a problem is specified by an input graph G with each edge assigned a cost. The goal is to output a subgraph of G satisfying certain connectivity requirements at minimum cost (measured as the sum of the costs of edges used). These requirements are described by specifying a minimum number of edges that must cross each cut of G. This formulation easily captures many classic problems including perfect matching, minimum cost flow, Steiner tree, and minimum T-join. By applying randomized rounding, we improve the approximation bounds for a large class of network design problems, from $O(\log n)$ (due to Goemans et al. [GGP+94]) to $1+o(1)$ in some cases. Our graph sampling theorems provide the necessary tools for showing that randomized rounding works well in this case.

We also apply randomized rounding to the classic *graph coloring problem*. No linear programs have yet been devised that provide a useful fractional solution, so we use more powerful *semidefinite programming* as our starting point. We show that any 3-colorable graph can be colored in polynomial time with $\tilde{O}(n^{1/4})$ colors, improving on the previous best bound of $\tilde{O}(n^{3/8})$ [Blu94]. We also give presently best results for k-colorable graphs. Along the way, we discover new properties of the *Lovász J-function*, an object that has received a great deal of attention because of its connections to graph coloring, cliques, and independent sets. This work is joint with Rajeev Motwani and Madhu Sudan [KMS98]. We gave a slight improvement with Avrim Blum [BK97].

2.4 Monte Carlo Estimation

The last randomization technique we consider is Monte Carlo estimation. The technique is applied when we want to estimate the probability p of a given event E over some probability space. Monte Carlo estimation carries out repeated "trials" (samples from the probability space) and measures in what fraction of the trials the event E occurs. This gives a natural estimate of the event probability.

The Monte Carlo approach breaks down when the interesting probability p is very small. To estimate p, we need to carry out enough experiments to see at least a few occurrences of E. But we expect to see a first occurrence only after $1/p$ trials, which may be too many to carry out efficiently. A solution to this problem, explored by Karp, Luby and Madras [KLM89], is to carry out the Monte Carlo simulation in a different, "biased" way that makes the event E more likely to occur, so that we can get by with fewer trials. The trick is to choose the new simulation so that it gives us useful information about the *original* probability space.

We apply this technique to the *network reliability problem*. In this problem, we are interested in estimating the probability that a network is disconnected by random edge failures, so it is perhaps unsurprising that randomization is useful. We are given a graph G whose edges fail randomly and independently with certain specified probabilities. Our goal is to determine the probability that the graph becomes disconnected by edge failures.

Unfortunately, it is known to be #*P*-hard (even worse than *NP*-hard) to exactly determine the reliability of a network. But we use Monte Carlo methods to give a *fully polynomial randomized approximation scheme (FPRAS)* for the network reliability problem [Kar98c]. Given a failure probability p for the edges, our algorithm, in time polynomial in n and $1/e$, returns a number P that estimates the probability FAIL(p) that the graph becomes disconnected. With high probability, P is in the range $(1\pm e)$FAIL(p). The algorithm is Monte Carlo, meaning that the approximation is correct with high probability but that it is not possible to verify its correctness. It generalizes to the case where the edge failure probabilities are different, to computing the probability the graph fails to be k-connected (for any fixed k), and to the more general problem of approximating the *Tutte Polynomial* for a large family of graphs. Our algorithm is easy to implement and appears likely to have satisfactory time bounds in practice [Kar98c, CGK+97, KT97].

A natural way to estimate a network's failure probability is to carry out numerous simulations of the edge failures and check how often the graph is disconnected by them. But as mentioned above, this can take prohibitively many trials if the failure probability is extremely small. However, we use our cut counting and sampling theorems to prove that when P is small, only the small cuts in a graph are significantly likely to fail. We use our cut algorithms to enumerate these small cuts and then use the biased Monte Carlo technique developed by Karp, Luby and Madras [KLM89] to estimate the probability that one of the explicitly enumerated cuts fails.

3 The Contraction Algorithm

To give some flavor of our results, we begin by describing an algorithm for finding a minimum cut in an undirected graph [KS96]. For simplicity we discuss unweighted graphs, but the algorithm works equally well for graphs with edge weights.

The algorithm is based on the idea of *contracting* edges. Suppose we were somehow able to identify an edge that did not cross the minimum cut. This would tell us that both of its endpoints are on the same side of the cut. We can use this information to simplify the graph by contracting the two endpoints. To contract two vertices v_1 and v_2 we replace them by a vertex v, and let the set of edges incident on v be the union of the sets of edges incident on v_1 and v_2. We do not merge edges from v_1 and v_2 that have the same other endpoint; instead, we allow multiple instances of those edges. However, we remove self loops formed by edges originally connecting v_1 to v_2. Formally, we delete all edges v_1,v_2, and replace each edge (v_1,w) or (v_2,w) with an edge (v,w). The rest of the graph remains unchanged. We will use $G/(v_1,v_2)$ to denote graph G with edge (v_1,v_2) contracted (by *contracting an edge*, we will mean contracting the two endpoints of the edge).

Note that a contraction reduces the number of graph vertices by one. We can imagine repeatedly selecting and contracting edges until every vertex has been merged into one of two remaining "metavertices." These metavertices define a cut of the original graph: each side corresponds to the vertices contained in one of the metavertices. It

is easy to see that if we never contract an edge that crosses the minimum cut, then the two metavertices we end up with will correspond to the two sides of the minimum cut we are looking for.

So our subgoal is to devise a method for selecting an edge that does not cross the minimum cut. There are some sophisticated deterministic algorithms for doing this [NI92], but unfortunately they are slow. We instead rely on the following observation: almost none of the edges in a graph cross the minimum cut. Thus, if we choose a *random* edge to contract, we probably get a non-min-cut edge! This gives us a very fast edge selection algorithm; the trade-off is that we must be prepared for it occasionally to make mistakes. We describe our algorithm in Figure 1. Assume initially that we are given a multigraph $G(V,E)$ with n vertices and m edges. The *Contraction Algorithm*, which is described in Figure 1, repeatedly chooses an edge at random and contracts it.

Algorithm Contract(G)
repeat until G has 2 vertices
choose an edge (v,w) uniformly at random from G
let $G \leftarrow G/(v,w)$
return the unique cut defined by (the contracted) G

Figure 1. The Contraction Algorithm

It is relatively straightforward to implement this algorithm in $O(n^2)$ time.

Lemma 3.1. *A particular minimum cut in G is returned by the Contraction Algorithm with probability at least $\binom{n}{2}^{-1}$.*

Proof. Fix attention on some specific minimum cut C with c crossing edges. We will use the term *minimum cut edge* to refer only to edges crossing C. If we never select a minimum cut edge during the Contraction Algorithm, then the two vertices we end up with must define the minimum cut.

Observe that after each contraction, the minimum cut value in the new graph must still be at least c. This is because every cut in the contracted graph corresponds to a cut of the same value in the original graph, and thus has value at least c. Furthermore, if we contract an edge (v,w) that does not cross C, then the cut C corresponds to a cut of value c in $G/(v,w)$; this corresponding cut is a minimum cut (of value c) in the contracted graph.

Each time we contract an edge, we reduce the number of vertices in the graph by one. Consider the stage in which the graph has r vertices. Since the contracted graph has a minimum cut of at least c, it must have minimum degree c, and thus at least $rc/2$ edges. However, only c of these edges are in the minimum cut. Thus, a randomly chosen edge is in the minimum cut with probability at most $2/r$. To determine the probability that we *never* contract a minimum cut edge, we simply multiply all of the per-stage probabilities. This shows that the probability that we never contract a minimum cut edge through all n-2 contractions is at least

$$\left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{3}\right) = \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdots\frac{2}{4}\cdot\frac{1}{3}$$
$$= \binom{n}{2}^{-1}$$
$$\geq 1/n^2.$$

□

Note that $\binom{n}{2}^{-1} \gg 1/n^2$, so the Contraction Algorithm described above has a relatively small chance of succeeding. But it is large enough to be useful. To improve our chance of success, we may simply repeat the algorithm a large number of times. If we run the Contraction Algorithm $n^2 \ln n$ times, and take the best answer we see, then the probability that we fail to give the right answer is just the probability that *none* of the repetitions of the algorithm yield the right answer, which is at most

$$(1-1/n^2)^{n^2 \ln n} \ll 1/n,$$

which means the algorithm works with high probability. This "amplification through repetition" is standard for randomized algorithms: we can get an exponential decrease in the failure probability from a linear slowdown in the running time.

Since the Contraction Algorithm takes $O(n^2)$ time per iteration, we immediately get an algorithm that finds a minimum cut *with high probability* in $O(n^4 \log n)$ time. This is somewhat unsatisfactory, as algorithms based on flow [HO94] can be used to find the minimum cut in $\tilde{O}(mn)$ time.

3.1 The Recursive Contraction Algorithm

By adding another idea we can improve the running time of our minimum cut algorithm to $\tilde{O}(n^2)$. We aim to "share work" among the numerous iterations of the algorithm. Note that the failure probability of the Contraction Algorithm rises as its size decreases. In fact, if we contract G until it has k vertices rather than 2, then the probability that the algorithm does not destroy the minimum cut of G exceeds $(k/n)^2$ (this follows by truncating the product we used to analyze the original Contraction Algorithm). So the real problem with the Contraction Algorithm arises when the graph has gotten small. We might imagine switching over to a deterministic algorithm once the graph is small, and indeed this approach yields improved performance. But we can do even better with another application of the principle that "repetition improves your chances." When the graph gets small, in order to improve our odds of success, we (recursively) carry out *two* executions of the algorithm on what remains.

Let Contract(G,k) denote a subroutine that runs the Contraction Algorithm until G is reduced to k vertices. Consider the *Recursive Contraction Algorithm* in Figure 2. As can be seen, we perform two independent trials. In each, we first partially contract the graph, but not so much that the likelihood of the cut surviving is too small. By contracting the graph until it has $n/\sqrt{2}$ vertices, we ensure a 50% probability of not contracting a minimum cut edge, so we expect that on the average one of the two attempts will avoid contracting a minimum cut edge. We then recursively apply the algorithm to each of the two partially contracted graphs. As described, the algorithm returns only a cut value; it can easily be modified to return a cut of the given value. Alternatively, we might want to output every cut encountered, hoping to enumerate all the minimum cuts.

Algorithm Recursive-Contract (G,n)
input A graph G of size n.
if G has 2 vertices
then
  return the weight of (unique) cut in G
else repeat twice
    $G' \leftarrow$ Contract ($G,n/\sqrt{2}$)
    Recursive-Contract ($G',n/\sqrt{2}$).
  return the smaller of the two resulting values.

Figure 2. The Recursive Contraction Algorithm

Next we analyze the running time of this algorithm.

Lemma 3.2. *Algorithm* Recursive-Contract runs in $O(n^2 \log n)$ *time.*

Proof. One level of recursion consists of two independent trials of contraction of G to $n/\sqrt{2}$ vertices followed by a recursive call. Performing a contraction to $n/\sqrt{2}$ vertices can be implemented by Algorithm Contract from the previous section in $O(n^2)$ time. We thus have the following recurrence for the running time:

$$T(n) = 2(n^2 + T(n/\sqrt{2})). \qquad (1)$$

This recurrence is solved by

$$T(n) = O(n^2 \log n).$$

□

We now analyze the probability that the algorithm finds the particular minimum cut we are looking for. We will say that the Recursive Contraction Algorithm *finds* a certain minimum cut if that minimum cut corresponds to one of the leaves in the algorithm's tree of recursive calls. Note that if the algorithm finds any minimum cut then it will output the minimum cut value.

Lemma 3.3. *The Recursive Contraction Algorithm finds a particular minimum cut with probability* $\Omega(1/\log n)$.

Proof. We give a recursive argument. The algorithm will find a particular minimum cut if, in one of its two iterations, the following two things happen: (i) the call to Contract ($G,n/\sqrt{2}$) preserves the minimum cut and (ii) the recursive call finds the particular minimum cut.

►

The probability that an iteration succeeds is just the product of the probabilities of events (i) and (ii). The algorithm succeeds if *either* iteration succeeds, and thus fails only if *both* iterations fail. The probability that this double failure happens is just the square of the probability that one iteration fails. Thus the success probability is one minus this squared quantity. This yields a recurrence $P(n)$ for a lower bound on the probability of success on a graph of size n:

$$P(2) = 1$$

$$P(n) \geq 1 - \left(1 - \frac{1}{2} P\left(n/\sqrt{2}\right)\right)^2$$

We solve this recurrence through a change of variables. Write $z_k = 4/P(2^{k/2}) - 1$, so $P(2^{k/2}) = 4/(z_k+1)$. Plugging this into the above recurrence and solving for z_k yields

$$z_1 = 3$$
$$z_{k+1} = z_k + 1 + 1/z_k.$$

Since clearly $z_k \geq 1$, it follows by induction that

$$k < z_k < 3 + 2_k$$

Thus $z_k = \Theta(k)$ and thus that

$$P(n) = 4/(z_{2\log n} + 1) = \Theta(1/\log n).$$

In other words, one trial of the Recursive Contraction Algorithm finds any particular minimum cut with probability $\Omega(1/\log n)$. □

Those familiar with branching processes might see that we are evaluating the probability that the extinction of contracted graphs containing the minimum cut does not occur before depth $2 \log n$.

Theorem 3.4 ([KS96]). *All minimum cuts in an arbitrarily weighted undirected graph with n vertices can be found with high probability in $O(n^2 \log^3 n)$ time.*

Proof. We will see below that there are at most $\binom{n}{2}$ minimum cuts in a graph. Repeating Recursive-Contract $O(\log^2 n)$ times gives an $O(1/n^4)$ chance of missing any particular minimum cut. Thus our chance of missing any one of the at most $\binom{n}{2}$ minimum cuts is upper bounded by $O(\binom{n}{2} \cdot n^{-4}) = O(1/n^2)$. □

3.2 Counting Cuts

Besides serving as an algorithm to find minimum cuts, the Contraction Algorithm tells us some interesting things about the number of minimum and, more generally, small cuts in a graph. These results are extremely useful when we consider our next topic, random sampling from graphs.

Definition 3.5. *An α-minimum cut is a cut whose value is at most α times that of the (global) minimum cut.*

Lemma 3.6. *There are at most $\binom{n}{2} < n^2$ minimum cuts.*

Proof. We showed that the Contraction Algorithm outputs a given minimum cut with probability at least $\binom{n}{2}^{-1}$. Suppose that there were more than k minimum cuts. Each is output with probability k. Since these output events are disjoint (the algorithm outputs only one cut), the probability that one of them is output is just the sum of their individual probabilities, namely $k/\binom{n}{2}$. This quantity, being a probability, is at most one. So $k \leq \binom{n}{2}$.

Theorem 3.7 (Cut Counting [KS96]). *In a graph with minimum cut c, there are less than $n^{2\alpha}$ cuts of value at most αc.*

Proof. If we consider a cut of value αc, we can prove (by generalizing the argument we gave for minimum cuts in the obvious way) that the Contraction Algorithm outputs it with probability at least $1/n^{2\alpha}$. The argument then proceeds as in the previous lemma.

4 Random Sampling

So far we have addressed random selection, which works by finding a "typical" element (e.g., a non-min-cut edge). We now turn to random sampling, where the goal is to build a small representative model of our input problem. We will describe algorithms for approximating and exactly finding maximum flows and minimum cuts in an undirected graph. For simplicity, we will restrict our discussion to graphs with unit-capacity edges (unweighted graphs) though many of the techniques that we discuss can be applied to weighted graphs as well. Due to space limitations, and because we are focusing on our thesis work rather than later improvements, we present algorithms that only work well when the minimum cut of the graph is large.

In unweighted graphs, the *s-t maximum flow problem* is to find a maximum set, or *packing*, of edge-disjoint s-t paths. It is known [FF62] that the value of this flow is equal to the value of the minimum s-t cut. In fact, the only known algorithms for finding an s-t minimum cut simply identify a cut that is saturated by an s-t maximum flow.

In unweighted graphs, a classic algorithm for finding such a maximum flow is the *augmenting path* algorithm (cf. [Tar83, AMO93]). Given a graph and an s-t flow of value f, a linear-time depth first search of the so-called *residual graph* will either show how to augment the flow to one of value $f+1$ or prove that f is the value of the maximum flow. This algorithm can be used to find a maximum flow of value v in $O(mv)$ time by finding v augmenting paths. Of course, since the algorithm's running time depends on the edge count and flow value, we can make it faster by reducing one or both quantities. We show how random sampling can be used to do this.

4.1 A Sampling Theorem

Our algorithms are all based upon the following model of random sampling in graphs. We are given an unweighted graph $G=(V,E)$ with a *sampling probability* p for each edge e, and we construct a random subgraph, or *skeleton*, on the same vertices V by placing each edge e in the skeleton independently with probability p. We denote the skeleton by $G(p)$. Note that if a given cut has k edges crossing it in G, then the expected number of edges crossing that cut in $G(p)$ is pk. In particular, if the s-t minimum cut in G has value v, then we might expect that the s-t minimum cut in $G(p)$ has value pv.

Unfortunately, samples invariably *deviate* from their expectations. In order to effectively make use of a skeleton, we need to show that these deviations are small. If they are, then the skeleton will tell us things about the original graph that are approximately correct. Let c be the minimum cut of graph G. Our main theorem says that so long as pc (the minimum expected cut value in the skeleton) is sufficiently large, every cut in the skeleton takes on roughly its expected value.

Theorem 4.1 ([Kar98b]).

Let $\varepsilon = \sqrt{3(d+2)(\ln n)/pc}$ (so $p = \Omega((\ln n)/\varepsilon^2 c)$). If $\varepsilon \leq 1$, then with probability $1 - O(1/n^d)$, every cut in $G(p)$ has value between $1-\varepsilon$ and $1+\varepsilon$ times its expected value.

This result is somewhat surprising. A graph has exponentially many (2^{n-1}) cuts. Naively, even if each cut is unlikely to deviate far from its expected value, with so many cuts one probably will. We are saved by the cut counting theorem discussed in the previous section. The central limit theorem (as quantified by the Chernoff bound [Che52, MR95b]) says that as the expected value of a sample gets larger, its sample value becomes more and more tightly concentrated about its expectation. In particular, as a cut value grows, its probability of deviating by a given ratio ε from its expectation decays exponentially with the cut value. The cut counting theorem says that the number of cuts of a given value increases "only" exponentially with the cut value. The parameters of Theorem 4.1 are chosen so that the exponential decrease in deviation probability dominates the exponential increase in the number of cuts.

4.2 Applications

We now show how the skeleton approach can be applied to minimum cuts and maximum flows. We use the following definitions:

Definition 4.2. *An α-minimum s-t cut is an s-t cut whose value is at most α times the value of the s-t minimum cut. An α-maximum s-t flow is an s-t flow whose value is at least α times the optimum.*

We have the following immediate extension of Theorem 4.1:

Theorem 4.3. *Let G be any graph with minimum cut c and let $p=\Omega((\ln n)/\varepsilon^2 c)$ as in Theorem 4.1. Suppose the s-t minimum cut of G has value v. Then with high probability, the s-t minimum cut in $G(p)$ has value between $(1-\varepsilon)pv$ and $(1+\varepsilon)pv$, and the minimum cut has value between $(1-\varepsilon)pc$ and $(1+\varepsilon)pc$.*

Corollary 4.4. *Assuming $\varepsilon<1/2$, the s-t min-cut in $G(p)$ corresponds to a $(1+4\varepsilon)$-minimum s-t cut in G.*

Proof. Assuming that Theorem 4.3 holds, the minimum cut in G is sampled to a cut of value at most $(1+\varepsilon)c$ in $G(p)$. So $G(p)$ has minimum cut no larger. And (again by the previous theorem) this minimum cut corresponds to a cut of value at most $(1+\varepsilon)c/(1-\varepsilon)<(1+4\varepsilon)c$ when $\varepsilon < 1/2$.

This means that if we use augmenting paths to find maximum flows in a skeleton, we find them faster than in the original graph for two reasons: the sampled graph has fewer edges, and the value of the maximum flow is smaller. The maximum flow in the skeleton reveals an s-t minimum cut in the skeleton, which corresponds to a near-minimum s-t cut of the original graph. An extension of this idea lets us find near-maximum flows: we randomly partition the graph's edges into many groups (each a skeleton), find maximum flows in each group, and then merge the skeleton flows into a flow in the original graph. Furthermore, once we have an approximately maximum flow, we can turn it into a maximum flow with a small number of augmenting path computations. This leads to an algorithm called DAUG that finds a maximum flow in $O(mv\sqrt{(\log n)/c})$ time, improving on the basic augmenting paths algorithm when c is large.

In the following subsections, we detail the algorithms we just sketched. We lead into DAUG with some more straightforward algorithms.

4.2.1 Approximate s-t Minimum Cuts

The most obvious application of Theorem 4.3 is to approximate s-t minimum cuts. We can find an approximate s-t minimum cut by finding an s-t minimum cut in a skeleton.

Lemma 4.5. *In a graph with minimum cut c, a $(1+\varepsilon)$-approximation to the s-t minimum cut of value v can be computed in $\tilde{O}(mv/\varepsilon^3c^2)$ time (with a low probability of error).*

Proof. Given ε, determine the corresponding $p=\Omega((\log n)/\varepsilon^2 c)$ from Theorem 4.3. Suppose we compute an s-t maximum flow in $G(p)$. By Theorem 4.3, $1/p$ times the value of the computed maximum flow gives a $(1+\varepsilon)$-approximation to the s-t min-cut value (with high probability). Furthermore, any flow-saturated (and thus s-t minimum) cut in $G(p)$ will be a $(1+\varepsilon)$-minimum s-t cut in G.

By the Chernoff bound [Che52, MR95b], the skeleton has $O(pm)$ edges (that is, about its expectation) with high probability. Also, by Theorem 4.3, the s-t minimum cut in the skeleton has value $O(pv)$. Therefore, the standard augmenting path algorithm can find a skeletal s-t maximum flow in $O((pm)(pv))= O(mv\log^2 n/\varepsilon^4c^2)$ time. Our improved augmenting paths algorithm DAUG in Section 4.2.4 lets us shave a factor of $\Omega(\sqrt{pc/\log n}) = \Omega(1/\varepsilon)$ from this running time, yielding the claimed bound. □

4.2.2 Approximate Maximum Flows

A slight variation on the previous algorithm will compute approximate maximum flows.

Lemma 4.6. *In a graph with minimum cut c and s-t maximum flow v, a $(1-\varepsilon)$-maximum s-t flow can be found in $\tilde{O}(mv/\varepsilon c)$ time (with a low probability of error).*

Proof. Given p as determined by ε, randomly partition the edges into $1/p$ groups, creating $1/p$ graphs. Each graph looks like (has the distribution of) a p-skeleton, and thus with high probability has an s-t minimum cut of value at least $pv(1-\varepsilon)$. It has an s-t maximum flow of the same value that can be computed in $O((pm)(pv))$ time as in the previous section (the skeletons are not independent, but even the sum of the probabilities that any one of them violates the sampling theorem is negligible). Adding the $1/p$ flows that result gives a flow of value $v(1-\varepsilon)$. The running time is $O((1/p)(pm)(pv))=O(mv(\log n)\varepsilon^2 c)$. If we use our improved augmenting path algorithm DAUG in Section 4.2.4, we improve the running time by an additional factor of $\Omega(1/\varepsilon)$, yielding the claimed bound.

4.2.3 A Las Vegas Algorithm

Our max-flow and min-cut approximation algorithms are both Monte Carlo, since they are not *guaranteed* to give the correct output (though the error probability can be made arbitrarily small). However, by combining the two approximation algorithms, we can certify the correctness of our results and obtain a *Las Vegas* algorithm for both problems–one that is guaranteed to find the right answer, but has a small probability of taking a long time to do so. This is a standard example of turning a Monte Carlo (error-prone) algorithm into a Las Vegas (correct but occasionally slow) one by checking the correctness of the output and trying again if it is wrong.

Corollary 4.7. *In a graph with minimum cut c and s-t maximum flow v, a $(1-\varepsilon)$-maximum s-t flow and a $(1+\varepsilon)$-minimum s-t cut can be found in $\tilde{O}(mv/\varepsilon c)$ time by a Las Vegas algorithm.*

Proof. Run both the approximate min-cut and approximate max-flow algorithms, obtaining (with high probability) a $(1-\varepsilon/2)$-maximum flow of value v_0 and a $(1+\varepsilon/2)$-minimum cut of value v_1. We know that $v_0 \le v \le v_1$, so to verify the correctness of the results all we need do is check that $(1+\varepsilon/2)v_0 \ge (1-\varepsilon/2)v_1$, which happens with high probability. To make the algorithm Las Vegas, we repeat both algorithms until each demonstrates the other's correctness (or switch to a deterministic algorithm if the first randomized attempt fails). We are right on the first try with high probability, so the algorithm runs fast with high probability. □

4.2.4 Exact Maximum Flows

We now use the above sampling ideas to speed up the familiar augmenting paths algorithm for maximum flows. This section is devoted to proving the following theorem:

Theorem 4.8 ([Kar98b]). *In a graph with minimum cut value c, a maximum flow of value v can be found in $\tilde{O}(mv/c)$ time by a Las Vegas algorithm.*

We assume for now that $v \ge \log n$. Our approach is a randomized divide-and-conquer algorithm that we analyze by treating each subproblem as a (non-independent) random sample. This technique gives a general approach to solving packing problems with an augmentation algorithm (including packing bases in a matroid [Kar93]). The flow that we are attempting to find can be seen as a packing of disjoint s-t paths. We use the algorithm in Figure 3, which we call DAUG (Divide-and-conquer AUGmentation).

1. Randomly split the edges of G into two groups (each edge goes to one or the other group with probability 1/2), yielding graphs G_1 and G_2.
2. Recursively compute s-t maximum flows in G_1 and G_2.
3. Add the two flows, yielding an s-t flow f in G.
4. Use augmenting paths to increase f to a maximum flow.

Figure 3. Algorithm DAUG

Note that we cannot apply sampling in DAUG's cleanup phase (Step 4) because the residual graph we manipulate there is directed, while our sampling theorems apply only to undirected graphs. We have left out

►

a condition for terminating the recursion; when the graph is sufficiently small (say with one edge) we use the basic augmenting path algorithm.

The outcome of Steps 1-3 is a flow. Regardless of its value, Step 4 will transform this flow into a maximum flow. Thus, our algorithm is clearly correct; the only question is how fast it runs. Suppose the *s*-*t* maximum flow is v. Consider G_1. Since each edge of G is in G_1 with probability 1/2, we can apply Theorem 4.3 to deduce that with high probability the *s*-*t* maximum flow in G_1 is at least $(v/2)(1 - \tilde{O}(\sqrt{1/c}))$ and the global minimum cut is $\Omega(c/2)$. The same holds for G_2 (the two graphs are not independent, but this is irrelevant). It follows that the flow f has value $v(1 - \tilde{O}(1/\sqrt{c})) = v - \tilde{O}(v/\sqrt{c})$. Therefore the number of augmentations that must be performed in G to make f a maximum flow is $\tilde{O}(v/c)$. Each augmentation takes $O(m)$ time on an m-edge graph. Intuitively, this suggests the following recurrence for the running time of the algorithm in terms of m, v, and c:

$$T(m,v,c) = 2T(m/2,v/2,c/2) + \tilde{O}(mv/c).$$

(where we use the fact that each of the two subproblems expects to contain $m/2$ edges). If we solve this recurrence, it evaluates to $T(m,v,c) = \tilde{O}(mv/c)$.

Unfortunately, this argument does not constitute a proof because the actual running time recurrence is in fact a *probabilistic recurrence*: the number of edges and sizes of cuts in the subproblems are random variables not guaranteed to equal their expectations. In particular, the recursion arguments is likely to be false when $c=o(\log n)$. Actually proving the result requires some additional work [Kar98b].

5 Randomized Rounding

Next we turn to Randomized Rounding. Randomized Rounding is a powerful method for approximately solving integer programming problems. The basic idea is to take the values of some *relaxation* of the problem (e.g., a linear program) and use them to generate integer values that define a solution to the integer program. There are two elements of a randomized rounding approach: a good relaxation that preserves much of the structure of the original intractable problem but can be solved efficiently, and a rounding strategy that transforms the relaxed solution into an integer one (along with a proof that it works well).

We apply randomized rounding to two *NP*-complete problems: network design and graph coloring. Both rounding approaches are slightly unusual. In the network design problem, we simultaneously round against exponentially many constraints. For graph coloring, we use *semidefinite programming* instead of the more traditional linear programming to determine a structure-preserving relaxation.

5.1 Network Design

The network design problem is a mirror to the minimum cut problem. The input is a set of vertices and a collection of candidate edges, each of which can be purchased for some specified cost. The goal is to design a network whose cuts are "sufficiently large." For example, one might wish to build (at minimum cost) a network that is k-connected. Alternatively one might want a network with sufficient capacity to route a certain amount of flow v between two vertices s and t (thus, the network must have *s*-*t* minimum cut v). Network design also covers many other classic problems, often *NP*-complete, including perfect matching, minimum cost flow, Steiner tree, and minimum T-join. A minimum cost 1-connected graph is just a minimum spanning tree, but for larger values of k the minimum-cost k-connected graph problem is *NP*-complete even when all edge costs are 1 or infinity [ET76].

Whenever a network design problem can be formulated in terms of (lower bound) constraints on the capacity or number of edges crossing each cut, one can write it as an integer linear program with a 0/1 variable for each edge that may be purchased and a constraint for each cut. To make the problem more tractable, we can relax the requirement that variables take 0/1 values and allow them to take fractional values in the interval [0,1]. This gives rise to a linear programming relaxation that can often be solved in polynomial time. Sometimes the linear programs can be represented compactly and solved with standard methods. At other times, even though the relaxation has exponentially many constraints, it has a good separation oracle (e.g. a minimum cut computation for the k-connected subgraph problem) and can thus be solved with the ellipsoid algorithm.

Solving the relaxation yields a fractional solution. Randomized rounding is used to convert the fractional solution back into an integral one. Given fractional variable values $x_1,\ldots,x_m$, we convert them to integer values $y_1,\ldots,y_m$ by setting $y_i=1$ with probability x_i and 0 otherwise. Note that $E[y_i]=x_i$. It follows that if $ax=b$ for some constraint vector a and scalar b, then $E[ay]=b$. In other words, y is "expected" to satisfy the same constraint that x did.

The problem, of course, is that random experiment deviate somewhat from their expectation. Raghavan and Thompson [RT87] showed that these deviations are often (provably) small enough that the resulting rounded solution is an approximately optimal solution to the integer program. Unfortunately, their analysis is focused on problems with a small number of constraints, which lets them argue that massive deviations from expectation are unlikely to happen. The network design problem has exponentially many constraints, so even unlikely large deviations are likely to occur in some of them. Fortunately, an analogue to our cut sampling theorem bounds these deviations, with the conclusion that randomized rounding can be applied to "fractional graphs" with much the same approximation guarantees as the original Raghavan-Thompson analysis. Among the results this yields is a $1+O((\log n)/k)$ approximation algorithm for the minimum k-connected subgraph problem [Kar98b].

5.2 Graph Coloring

We also apply randomized rounding to the problem of graph coloring. This problem is *NP*-complete and has recently been proven extremely hard even to approximate well on graphs with large chromatic number [LY93]. However, there still remains some hope that it might be possible to do reasonably well coloring a graph with small chromatic number. In our thesis, we focus on 3-colorable graphs, and show how to color them with $\tilde{O}(n^{1/4})$ colors. The technique extends to give new performance ratios for graphs with larger chromatic number. This work is joint with Rajeev Motwani and Madhu Sudan [KMS98] and built upon the exciting work of Goemans and Williamson [GW95] on the maximum cut problem. We later improved it in joint work with Avrim Blum [BK97].

To attack graph coloring, we turned to the recently developed technique of *semidefinite programming*. Instead of rounding fractional-valued scalars to integers, we round *vectors*. To illustrate, we describe the relaxation of our graph coloring problem. We aim to assign a unit-length *vector* v_i to each vertex i of our graph such that for any two adjacent vertices i and j, the dot product $v_i \cdot v_j \le -1/2$. To see that this can be done to any three-colorable graph, consider a "star" of three vectors on the unit circle with 120° angles between them, for example (1,0), (-1/2, $\sqrt{3}$/2), and (-1/2, -$\sqrt{3}$/2). Each has unit length and has dot product -1/2 with the other two vectors. Given a 3-colored graph, we can solve the vector problem by assigning the first vector to all red vertices, the second to all green, and the third to all blue vertices. This proves that any 3-colorable graph has a feasible solution to our vector problem, which means that it is a valid relaxation.

Solving the relaxation can be formulated as finding a feasible (vector) solution to the following *semidefinite program* (where E denotes the set of edges in the graph G).

$$v_i \cdot v_j \le -1/2 \text{ if } (i,j) \in E$$
$$v_i \cdot v_i = 1.$$

The fact that such a system of constraints (on any linear combination of dot products) can be solved (to within a negligibly small error) in polynomial time is a difficult result [GLS88] which we can fortunately use as a black box.

Unfortunately, there are many feasible assignments to this semidefinite program–most in a dimension much higher than 2. We cannot constrain the solution to be

two dimensional (and still solve the problem in polynomial time) so we must decide how to take a high dimensional relaxed solution and transform it into a coloring. Our method for doing so is quite straightforward: we choose a number of *random* unit vectors as *centers*, and color vertex i with the center closest to v_i. We show that if the number of centers is sufficiently large, no two adjacent vertices are likely to be assigned to the same center–that is, we get a legal coloring. The intuition behind our argument is simple. The vectors for adjacent vertices i and j point "away" from each other thanks to the semidefinite constraints. Thus, if i is "near" a random center, j will be "far" from that center and is thus likely to end up attached to some other center. Some technical arguments involving Gaussian distributions suffice to prove that $\tilde{O}(n^{1/4})$ centers suffice to make the probabilities work out.

6 Monte Carlo Estimation

The last randomization technique we consider is Monte Carlo estimation. The technique is applied when we want to estimate the probability p of a given event over some probability space. Monte Carlo estimation carries out repeated "trials" (samples from the probability space) and measures how often the given event occurs. This gives a natural estimate of the event probability.

We use Monte-Carlo estimation to attack the *all-terminal network reliability problem*: given a network on n vertices, each of whose m links is assumed to fail (disappear) independently with some probability, determine the probability that the surviving network is connected. The practical applications of this question to communication networks are obvious, and the problem has therefore been the subject of a great deal of study. A comprehensive survey can be found in [Col87]. As mentioned in Section 2, this problem is *#P*-hard to solve exactly, so we give a *fully polynomial randomized approximation scheme (FPRAS)* that gives an answer accurate to within a relative error of ε in time polynomial in n and $1/\varepsilon$. Although our algorithm is quite general [Kar98c], we restrict discussion here to the case where every edge fails independently with the same probability p. We let FAIL(p) denote the failure probability of G when edges fail with probability p.

The basic approach of our FPRAS is to consider two cases. When FAIL(p) is large, we estimate it in polynomial time by direct Monte Carlo simulation of edge failures. That is, we randomly fail edges and check whether the graph remains connected. Since FAIL(p) is large, a small number of trials gives enough data to estimate it well. When FAIL(p) is small, we show that we can focus on the small cuts in a graph. We enumerate them with our cut algorithms and then use a biased Monte Carlo estimation technique to determine their failure probability.

Observe that a graph becomes disconnected precisely when all of the edges in some cut of the graph fail. If each edge fails with probability p, then the probability that a k-edge cut fails is p^k. Thus, the smaller a cut, the more likely it is to fail. It is therefore natural to focus attention on the small graph cuts. In particular, the probability that the graph becomes disconnected is at least p^c (since this is the probability that a minimum cut fails). At the same time, the probability that any one α-minimum cut fails is $p^{\alpha c}$.

We can now describe our two cases. When FAIL(p) $\ddagger p^c \ddagger n^{-3}$, we use direct Monte Carlo simulation to estimate the failure probability. A single experiment consists of flipping coins to see which edges fail and then checking whether the graph is connected. If we carry out roughly $(\log n)/\varepsilon^2$ FAIL(p)$=\tilde{O}(n^3/\varepsilon^2)$ experiments (a polynomial number), we will see about $(\log n)/\varepsilon^2$ failures. This provides enough "evidence" to give a good estimate of the failure probability [Che52, KLM89].

Unfortunately, when FAIL(p) is small, we need too many simulations to develop a good baseline (note that we do not expect to see a single failure until we perform 1/FAIL(p) experiments; this number can be super-polynomial). We instead turn to an enumeration of the small cuts. When $p^c \pounds n^{-3}$, we know that a given α-minimum cut fails with probability $p^{\alpha c} \pounds n^{-3\alpha}$. But we argued in our Cut Counting theorem that the number of α-minimum cuts is only $n^{2\alpha}$. It follows that the probability that *any* α-minimum cut fails is less than $n^{-\alpha}$–that is, exponentially decreasing with α. Thus, for a relatively small α, the probability that a greater than α-minimum cut fails is negligible. We can therefore approximate FAIL(p) by approximating the probability that some less than α-minimum cut fails. We do so by enumerating the α-minimum cuts (using a modification of the Contraction Algorithm [KS96]) and then applying a *DNF counting* algorithm developed by Karp, Luby, and Madras [KLM89]. The algorithm of [KLM89] is also based on Monte-Carlo methods, but uses biased sampling to ensure that we see failures often so that a good estimate of their likelihood can be constructed quickly. The contribution of our work is to show that it is possible to build a small formula that can be fed to the DNF counting algorithm to produce a meaningful answer.

7 Conclusion

Randomization has become an essential tool in the design of optimization algorithms. Randomization leads to algorithms that are faster, simpler, and/or better-performing than their deterministic counterparts. The basic techniques of random selection, random sampling, randomized rounding and Monte Carlo estimation let us draw on our intuitions about common cases and representative samples: whenever we expect that something should "usually" happen or be "typical," randomization may give us a way to turn our suspicion into an algorithm. We have demonstrated this approach on numerous basic optimization problems. But a great deal of work remains to be done.

The most direct open question is how far our particular results can be pushed. The minimum spanning tree and minimum cut problems are essentially "done," one with a linear time algorithm and the other with a linear-times-polylog time algorithm; but our results on s-t minimum cuts and maximum flows seem very incomplete: no lower bounds are evident, and our upper bounds are "odd" (e.g. $\tilde{O}(n^{20/9})$ for flows in simple graphs [KL98]) in ways that suggest that it must be possible to improve them (e.g. to $O(n^2)$). Our approximation algorithms apply well to both capacitated and uncapacitated problems, but our exact algorithms so far apply best to uncapacitated problems. We suspect that more can be done here.

More questionable is whether any of our technology can be applied to *directed* graphs. Absolutely none of the results discussed in this article extend to directed graphs: the Contraction Algorithm fails on them, and as a result we have been unable to prove a sampling theorem, a cut counting theorem (in fact a directed graph can have exponentially many minimum cuts), a sampling theorem, a rounding theorem, or anything about directed reliability. One possible explanation for this is that undirected graphs form natural matroids while directed graphs do not [Kar93].

Thinking more broadly, a fundamental question about randomization is whether it is truly "necessary." Often, after a randomized algorithm gives insight into a problem, one can devise a deterministic algorithm with some of the same properties. Within theoretical computer science, there is an entire subfield devoted to *derandomization* – the development of techniques that will mechanically convert a randomized algorithm into a deterministic one. For example, the randomized rounding procedure for (polynomial size) linear programs can be made deterministic [Rag88], as can our randomized rounding algorithm for graph coloring [MR95a]. We have also derandomized our Contraction Algorithm [KM97].

Even when it is possible to derandomize an algorithm, it may not be worth doing so. The derandomization can add complexity, either computational (e.g. in the case of the Contraction Algorithm, where the derandomization drastically slows the algorithm) or conceptual (e.g. for randomized rounding, where the intuitive expectation argument is replaced by a more complex numeric calculation).

However, there are still motivations for exploring the derandomization question. Perhaps the strongest is the wish for an algorithm with predictable behavior. In a situation with lives at stake, it would be unsatisfactory to be right *most* of the time, or *usually* fast enough. This

►

problem is particularly acute with our Monte Carlo algorithms, where one cannot even tell whether the answer is correct! An obvious place to begin is the minimum cut problem, where a Monte Carlo algorithm can solve the problem (with high probability) in $\tilde{O}(m)$ time but the best known deterministic running time is $\tilde{O}(mn)$. Another specific question is whether there is a *deterministic* linear-time minimum spanning tree algorithm (which would finally put the problem to rest for good). A more abstract question is the following: we have proven that any graph has a sparse "skeleton" that accurately approximates its cuts; this seems to have assorted uses. Can such a skeleton be constructed deterministically in polynomial time?

Comments and questions on this survey are most welcome.

–DAVID R. KARGER

References

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[BK96] A.A. Benczúr and D.R. Karger. Approximate *s-t* min-cuts in $\tilde{O}(n^2)$ time. In G. Miller, editor, *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 47-55. ACM, ACM Press, May 1996.

[BK97] A. Blum and D.R. Karger. Improved approximation for graph coloring. *Information Processing Letters*, 61(1):49-53, January 1997.

[BK98] A.A. Benczúr and D.R. Karger. Augmenting undirected edge connectivity in $\tilde{O}(n^2)$ time. In H. Karloff, editor, *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 500-509. ACM-SIAM, January 1998.

[Blu94] A. Blum. New approximation algorithms for graph coloring. *Journal of the ACM*, 41(3):470-516, May 1994.

[CGK+97] C.C. Chekuri, A.V. Goldberg, D.R. Karger, M.S. Levine, and C. Stein. Experimental study of minimum cut algorithms. In M. Saks, editor, *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 324-333. ACM-SIAM, January 1997.

[Che52] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493-509, 1952.

[Col87] C.J. Colbourn. *The Combinatorics of Network Reliability*, volume 4 of *The International Series of Monographs on Computer Science*. Oxford University Press, 1987.

[ET76] K.P. Eswaran and R.E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5:653-665, 1976.

[FF62] L.R. Ford Jr. and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[FR75] R.W. Floyd and R.L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165-172, 1975.

[GGP+94] M.X. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, É. Tardos, and D. Williamson. Improved approximation algorithms for network design problems. In D.D. Sleator, editor, *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223-232. ACM-SIAM, January 1994.

[GGST86] H.N. Gabow, Z. Galil, T.H. Spencer, and R.E. Tarjan. Efficient algorithms for finding minimum spanning tree in undirected and directed graphs. *Combinatorica*, 6:109-122, 1986.

[GLS88] M. Gröetschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, 1988.

[GR97] A. Goldberg and S. Rao. Beyond the flow decomposition barrier. In *Proceedings of the 30th Annual Symposium on the Foundations of Computer Science*, pages 2-11. IEEE, IEEE Computer Society Press, October 1997.

[GT88] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921-940, 1988.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 1995.

[HO94] Hao and Orlin. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17(3):424-446, 1994. A preliminary version appeared in Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms.

[Hoa62] C.A.R. Hoare. Quicksort. *Computer Journal*, 5(1):10-15, 1962.

[Kar93] D.R. Karger. Random sampling in matroids, with applications to graph connectivity and minimum spanning trees. In L. Guibas, editor, *Proceedings of the 34th Annual Symposium on the Foundations of Computer Science*, pages 84-93. IEEE, IEEE Computer Society Press, November 1993. To appear in Mathematical Programming B.

[Kar94] D.R. Karger. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, Stanford, CA 94305, 1994. Contact at karger@lcs.mit.edu. Available from http://theory.lcs.mit.edu/~karger.

[Kar96] D.R. Karger. Minimum cuts in near-linear time. In G. Miller, editor, *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 56-63. ACM, ACM Press, May 1996.

[Kar98a] D.R. Karger. Better random sampling algorithms for flows in undirected graphs. In H. Karloff, editor, *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 490-499. ACM-SIAM, January 1998.

[Kar98b] D.R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 1998. To appear. A preliminary version appeared in the 1994 Symposium on the Theory of Computing.

[Kar98c] D.R. Karger. A randomized fully polynomial approximation scheme for the all terminal network reliability problem. *SIAM Journal on Computing*, 1998. To appear. A preliminary version appeared in the 1995 Symposium on the Theory of Computing.

[KKT95] D.R. Karger, P.N. Klein, and R.E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321-328, 1995.

[KL98] D.R. Karger and M. Levine. Finding maximum flows in simple undirected graphs seems faster than bipartite matching. In *Proceedings of the 29th ACM Symposium on Theory of Computing*. ACM, ACM Press, May 1998.

[KLM89] R.M. Karp, M. Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429-448, September 1989.

[KM97] D.R. Karger and R. Motwani. Derandomization through approximation: An *NC* algorithm for minimum cuts. *SIAM Journal on Computing*, 26(1):255-272, 1997. A preliminary version appeared in STOC 1993, p. 497.

[KMS98] D.R. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2):246-265, March 1998.

[KS96] D.R. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601-640, July 1996. Preliminary portions appeared in SODA 1992 and STOC 1993.

[KT97] D.R. Karger and R.P. Tai. Implementing a fully polynomial time approximation scheme for all terminal network reliability. In M. Saks, editor, *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 334-343. ACM-SIAM, January 1997.

[LY93] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In A. Aggarwal, editor, *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 286-293. ACM, ACM Press, May 1993.

[MR95a] S. Mahajan and H. Ramesh. Derandomizing semidefinite programming based approximation algorithms. In *Proceedings of the 36th Annual Symposium on the Foundations of Computer Science*, pages 162-169. IEEE, IEEE Computer Society Press, October 1995.

[MR95b] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, 1995.

[NI92] H. Nagamochi and T. Ibaraki. Linear time algorithms for finding *k*-edge connected and *k*-node connected spanning subgraphs. *Algorithmica*, 7:583-596, 1992.

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximate packing integer programs. *Journal of Computer and System Sciences*, 37(2):130-43, October 1988.

[RT87] P. Raghavan and C.D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365-374, 1987.

[Tar83] R.E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 1983.

Solving Optimization Problems with AMPL and NEOS

In the conventional approach to solving optimization problems, the user must first identify and obtain the appropriate piece of optimization software; write code to define the problem in the manner required; and then compile, link, and run the software. For nonlinear problems, Fortran or C code must be written to define the problem, compute function values and derivatives, and specify sparsity patterns.

The Network-Enabled Optimization System (NEOS) Server at Argonne National Laboratory (ANL) provides a novel alternative to this process. NEOS allows users to solve optimization problems over the Internet with state-of-the-art software without downloading and linking code. NEOS provides the user with a choice of solvers. Once a nonlinear optimization problem is specified, NEOS uses automatic differentiation tools to compute derivatives and sparsity patterns, compiles all subroutines, links with the appropriate libraries, and executes the solver. The process is simplified for linear problems since there is no need to compute derivatives. In all cases, the user is given a solution, along with runtime statistics. Most of the computation is done in the background, hidden from view.

Although problem-solving is greatly simplified with NEOS, the user still needs to define the problem in Fortran or C. For many problems, this can require considerable effort. To streamline the process, Jorge Moré, in cooperation with leading experts in optimization software, added a series of AMPL solvers to the NEOS system. AMPL is a comprehensive and powerful modeling language for optimization problems that simplifies the formulation of optimization problems. AMPL's flexibility and convenience makes it suitable for rapid prototyping and model development.

The following solvers can be used for AMPL-formatted optimization problems on the NEOS Server: DONLP2, LANCELOT, LOQO, MINOS, and SNOPT. Each solver has instructions on how to submit a problem using AMPL, as well as pointers to a user guide and a set of sample problems.

Using the service is simple. The user only needs to provide the problem in AMPL format. The problem is specified by a model file, and optionally, a data file and command file. See http://www.netlib.org/ampl/models for examples of models in AMPL format. For the AMPL-PRO interface, which allows users to submit a problem to all the solvers, see http://www.mcs.anl.gov/otc/Server/neos/software-library/AMPL:AMPL-PRO.

Sample problems can be found at http://www.mcs.anl.gov/otc/Server/neos/software-library/AMPL:AMPL-PRO/trial.html.

Contributors to the development and installation of AMPL Solvers on the NEOS Server and the related web sites are as follows:

AMPL: David Gay, Bell Laboratories; and Bob Fourer, Northwestern University (http://www.ampl.com/cm/cs/what/ampl).

AMPL-PRO: Chih-Jen Lin, ANL and the University of Michigan (http://www.mcs.anl.gov/otc/Server/neos/software-library/AMPL:AMPL-PRO).

DONLP2: Hans Mittelmann, Arizona State University; and Peter Spellucci, Technical University of Darmstadt, Germany (http://www.mcs.anl.gov/otc/Server/neos/software-library/AMPL:DONLP2).

LANCELOT: Nick Gould, Rutherford Appleton Laboratories, UK (http://www.mcs.anl.gov/otc/Server/neos/software-library/AMPL:LANCELOT).

LOQO: Bob Vanderbei, Princeton University (http://www.mcs.anl.gov/otc/Server/neos/software-library/AMPL:LOQO).

MINOS: Michael Saunders, Stanford University (http://www.mcs.anl.gov/otc/Server/neos/software-library/AMPL:MINOS).

SNOPT: Philip Gill, University of California at San Diego (http://www.mcs.anl.gov/otc/Server/neos/software-library/AMPL:SNOPT).

For general information about the NEOS Server, including frequently asked questions, client software, submission tools, usage statistics, and collaborators, see http://www.mcs.anl.gov/otc/Server/.

– JORGE MORÉ

13509-city TSP solved

David Applegate, Robert Bixby, Vasek Chvátal, and Bill Cook have solved instance "usa13509" from TSPLIB. At the time the instance was added to TSPLIB (early in 1995), it consisted of the continental US cities having population 500 or more. The final computations were run on a cluster of 3 AlphaServer 4100's (12 processors total) and a network of 32 Pentium II PCs.

– MARY BETH HRIBAR

An Updated Mixed Integer Programming Library: MIPLIB 3.0

Robert E. Bixby
bixby@caam.rice.edu
Sebastián Ceria
sebas@cumparsita.gsb.columbia.edu
Cassandra M. McZeal
cmoore@caam.rice.edu
Martin W.P. Savelsbergh
Martin.Savelsbergh@isye.gatech.edu

1 Introduction

In response to the needs of researchers for access to challenging mixed integer programs, Bixby et al. [1] created MIPLIB, an electronically available library of both pure and mixed integer programs, most of which arise from real-world applications.

Since its introduction, MIPLIB has become a standard test set for comparing the performance of mixed integer optimization codes. Its availability has provided an important stimulus for researchers in this very active area. As technology has progressed, however, there have been significant improvements in state-of-the-art optimizers and computing machinery. Consequently, several instances have become too easy, and a need has emerged for more difficult instances. Also, it has been observed that certain types of problems are overrepresented in MIPLIB and others underrepresented. These considerations have prompted the present update.

Since mixed integer programming is such an active research area, and the performance of optimizers keeps improving, we anticipate that this update will not be the last. Subsequent updates are planned on a yearly basis. We encourage both researchers and practitioners in integer programming to submit real-world instances for consideration and possible inclusion in MIPLIB.

This note describes the MIPLIB update. We have added several new problems and deleted some existing ones. In addition, we have included, for each problem, certain auxiliary information describing the structure of the constraint matrix. The purpose of this information is to identify constraint classes that may be useful in the various phases of problem solving, such as preprocessing, constraint generation, and branching.

Table 1: Problem Statistics

NAME	ROWS	COLS	INT	0/1	CONT	INT SOLN	LP SOLN
10teams	230	2025	1800	ALL	225	924	917
air03	124	10757	10757	ALL	0	340160	338864.25
air04	823	8904	8904	ALL	0	56137	55535.436
air05	426	7195	7195	ALL	0	26374	25877.609
arki001	1048	1388	538	415	850	7580813.0459	7579599.80787
bell3a	123	133	71	39	62	878430.32	862578.64
bell5	91	104	58	30	46	8966406.49	8608417.95
blend2	274	353	264	231	89	7.598985	6.9156751140
cap6000	2176	6000	6000	ALL	0	-2451377	-2451537.325
dano3mip	3202	13873	552	ALL	13321	728.1111	576.23162474
danoint	664	521	56	ALL	465	65.67	62.637280418
dcmulti	290	548	75	ALL	473	188182	183975.5397
dsbmip	1182	1886	192	160	1694	-305.19817501	-305.19817501
egout	98	141	55	ALL	86	568.101	149.589
enigma	21	100	100	ALL	0	0.0	0.0
fast0507	507	63009	63009	ALL	0	174	172.14556668
fiber	363	1298	1254	ALL	44	405935.18000	156082.51759
fixnet6	478	878	378	ALL	500	3983	1200.88
flugpl	18	18	11	0	7	1201500	1167185.73
gen	780	870	150	144	720	112313	112130.0
gesa2	1392	1224	408	240	816	25779856.372	25476489.678
gesa2_o	1248	1224	720	384	504	25779856.372	25476489.678
gesa3	1368	1152	384	216	768	27991042.648	27833632.451
gesa3_o	1224	1152	672	336	480	27991042.648	27833632.451
gt2	29	188	188	24	0	21166.000	13460.233074
harp2	112	2993	2993	ALL	0	-73899798.00	-74353341.502
khb05250	101	1350	24	ALL	1326	106940226	95919464.0
l152lav	97	1989	1989	ALL	0	4722	4656.36
lseu	28	89	89	ALL	0	1120	834.68
misc03	96	160	159	ALL	1	3360	1910.0
misc06	820	1808	112	ALL	1696	12850.8607	12841.69
misc07	212	260	259	ALL	1	2810	1415.0
mitre	2054	10724	10724	ALL	0	115155	114740.51848
mod008	6	319	319	ALL	0	307	290.93
mod010	146	2655	2655	ALL	0	6548	6532.08
mod011	4480	10958	96	ALL	10862	-54558535	-62121982.552
modglob	291	422	98	ALL	324	20740508	20430947.0
noswot	182	128	100	75	28	-41	-43.0
nw04	36	87482	87482	ALL	0	16862	16310.66667
p0033	16	33	33	ALL	0	3089	2520.57
p0201	133	201	201	ALL	0	7615	6875.0
p0282	241	282	282	ALL	0	258411	176867.50
p0548	176	548	548	ALL	0	8691	315.29
p2756	755	2756	2756	ALL	0	3124	2688.75
pk1	45	86	55	ALL	31	11.0	0.0
pp08a	136	240	64	ALL	176	7350.0	2748.3452381
pp08aCUTS	246	240	64	ALL	176	7350.0	5480.6061563
qiu	1192	840	48	ALL	792	-132.873137	-931.638857
qnet1	503	1541	1417	1288	124	16029.692681	14274.102667
qnet1_o	456	1541	1417	1288	124	16029.692681	12095.571667
rentacar	6803	9557	55	ALL	9502	30356761	28806137.644
rgn	24	180	100	ALL	80	82.1999	48.7999
rout	291	556	315	300	241	1077.56	981.86428571
set1ch	492	712	240	ALL	472	54537.75	32007.73
seymour	4944	1372	1372	ALL	0	423	403.84647413
stein27	118	27	27	ALL	0	18	13.0
stein45	331	45	45	ALL	0	30	22.0
vpm1	234	378	168	ALL	210	20	15.4167
vpm2	234	378	168	ALL	210	13.75	9.8892645972

2 Problems

2.1 Problem Additions

Twenty-three new problems have been added to MIPLIB: *10teams, arki001, blend2, dano3mip, danoint, fast0507, fiber, gesa2, gesa2_o, gesa3, gesa3_o, gt2, harp2, mitre, nw04, pp08a,pp08aCUTS, pk1, qnet1, qnet1_o, rout, seymour, and vpm2*. Most of the new problems arose from real-world applications.

The *10teams* model is a sports scheduling model, describing the allocation of teams to time slots in the English football league. The primary objective is feasibility. The problem *arki001* describes raw-material extraction in batches to fit production processes. The problems *danoint* and *dano3mip* resulted from telecommunications applications. Problem *dano3mip* deals with ATM network layout (design). The application is described in more detail in "Computational experience with a difficult mixed-integer multi-commodity flow problem," by O. Günlük and D. Bienstock, which appeared in Mathematical Programming, 68 (1995), pp. 213-238.

The *fast0507* model is a set covering problem arising from a crew-scheduling application for a railway company. The model *fiber* is a fiber routing problem. The models *gesa2_o* and *gesa3_o* are general integer programs arising from the optimization of electricity generation in the Balearic Islands in Spain. The problems *gesa2* and *gesa3* are the same as for the "*_o*" versions, except that surrogate knapsacks, linking resources at the different islands, have been added. The LP and IP values are the same as the "*_o*" problems, but as integer problems they appear to be easier to solve.

The model *gt2* is a general integer program arising from a routing application, where the objective is to determine the number of trucks of a certain type that will be needed to satisfy "client demands." The problem *nw04* arises from an airline set partitioning application. The model *pp08aCUTS* is the same as *pp08a* except that flow cover inequalities have been added to strengthen the formulation. The problems *qnet* and *qnet_o* are line-leasing models. The problem *seymour* is a set-covering problem that arose from work related to the proof of the well-known 4-color Theorem from graph theory. It tries to find a "best" unavoidable set of reducible configurations. To our knowledge, this model has not been solved to provable optimality. The model *vpm2* is a variation of the *vpm1* model already in MIPLIB.

2.2 Problem Statistics

The following tables give various statistics for the complete, revised set of MIPLIB problems.

Table 2: Problem Origins

NAME	ORIGINATOR	FORMULATOR	DONATOR
10teams	Dash Associates		Bob Daniel
air03			Greg Astfalk
air04			Greg Astfalk
air05			Greg Astfalk
arki001	Avesta-Sheffield, Sweden	Nils Holmberg	Arne Stolbjerg Drud
bell3a	William Cook	William Cook	William Cook
bell5	William Cook	William Cook	William Cook
blend2	Dash Associates		Bob Daniel
cap6000	Karla Hoffman, Manfred Padberg	Telecommunications Corporation	Karla Hoffman
dano3mip	Bell Communications Research		Daniel Bienstock
danoint	Columbia's Center for Telecommunications Research		Daniel Bienstock
dcmulti	Jeremy Shapiro	Jeremy Shapiro	Jonathan Eckstein
dsbmip			John J. Forrest
egout	Etienne Loute	Laurence A. Wolsey	Martin W.P. Savelsbergh
enigma	Harlan Crowder	Harlan Crowder	E. Andrew Boyd
fast0507	Italian Railway Company	Pier Luigi Guida	Sebastián Ceria
fiber	US West	Youngho Lee	Martin W.P. Savelsbergh
fixnet6		T.J. Van Roy	Martin W.P. Savelsbergh
flugpl	Harvey M. Wagner	John W. Gregory	E. Andrew Boyd
gesa2	Spanish Electricity	Laurence A. Wolsey	Sebastián Ceria
gesa2_o	Spanish Electricity	GESA (Consulting Company)	Sebastián Ceria
gesa3	Spanish Electricity	Laurence A. Wolsey	Sebastián Ceria
gesa3_o	Spanish Electricity	GESA (Consulting Company)	Sebastián Ceria
gen		Laurence A. Wolsey	Martin W.P. Savelsbergh
gt2			Sebastián Ceria
harp2			Martin W.P. Savelsbergh
khb05250	Kuhn-Hamburger	Laurence A. Wolsey	Martin W.P. Savelsbergh
l152lav	Harlan Crowder	Harlan Crowder	John W. Gregory
lseu	C. E. Lemke, K. Spielberg	Ellis L. Johnson, Uwe H. Suhl	John J. Forrest
misc03			Greg Astfalk
misc06			Greg Astfalk
misc07			Greg Astfalk
mitre			Martin W.P. Savelsbergh
mod008	IBM France	IBM France	John J. Forrest
mod010	IBM Yorktown Hts	IBM Yorktown Hts	John J. Forrest
mod011	Uwe H. Suhl	Uwe H. Suhl	John J. Forrest
modglob	Y. Smeers	Laurence A. Wolsey	Martin W.P. Savelsbergh
noswot		Linus E. Schrage	John W. Gregory
nw04	Northwest Airlines		Karla Hoffman
p0033	CJP set		E. Andrew Boyd
p0201	CJP set		E. Andrew Boyd
p0282	CJP set		E. Andrew Boyd
p0548	CJP set	Ellis L. Johnson	E. Andrew Boyd
p2756	CJP set	Ellis L. Johnson	E. Andrew Boyd
pk1		Pinar Keskinocak	Sebastián Ceria
pp08a			Martin W.P. Savelsbergh
pp08aCUTS			Martin W.P. Savelsbergh
qiu	Yu-Ping Chiu	Yu-Ping Chiu	Jonathan Eckstein
qnet1	BASF	Laurence A. Wolsey	Sebastián Ceria
qnet1_o	BASF	BASF	Sebastián Ceria
rentacar			John J. Forrest
rgn	Linus E. Schrage	Laurence A. Wolsey	Martin W.P. Savelsbergh
rout	S. Graves	Hernan Abeledo	Sebastián Ceria
set1ch		Laurence A. Wolsey	Martin W.P. Savelsbergh
seymour			Paul Seymour
stein27	George L. Nemhauser	John W. Gregory	E. Andrew Boyd
stein45	George L. Nemhauser	John W. Gregory	E. Andrew Boyd
vpm1		Laurence A. Wolsey	Martin W.P. Savelsbergh
vpm2		Laurence A. Wolsey	Martin W.P. Savelsbergh

In Table 1, the first column, NAME, contains the name of the model. The next two columns, ROWS and COLS, contain the number of rows (constraints), not including free rows, and the number of columns (variables) in the problem, respectively. The column INT specifies the number of variables that are restricted to integer values, and the 0/1 column specifies how many of these integer variables are binary. The column CONT specifies the number of variables that are continuous. The next two columns report the best-known integral solution and the optimal value with the integrality restrictions relaxed, respectively. Three entries in the INT SOLN column include the qualifier (not opt) which indicates that the reported solution has not been proved to be optimal. The last column LP SOLN contains the solution to the linear programming relaxation for each problem.

Table 2 contains information regarding the origins of each problem. The first column is, again, NAME. The second column, ORIGINATOR, gives the name of the person (institution) from whom (which) the problem originated. The next column, FORMULATOR, gives the name of the person or organization responsible for formulating the model, and the last column, DONATOR, gives the name of the person or institution who contributed the problem.

2.3 Problem Deletions

As a part of the modifications to MIPLIB, 25 problems were deleted from the test set. One of the factors used to determine whether a problem was to be deleted was the ease with which the problem was solved. The remainder of the deleted problems were chosen to reduce duplication. These problems were each part of a set of problems exhibiting either similar form or behavior.

In order to determine which problems were "easy," all of the current MIPLIB models were run on a SPARC 10/41 using the CPLEX 3.0 mixed-integer optimizer[1]. The settings given below were used. They were meant to correspond to what one might consider the most straightforward implementation of a reasonable, LP-based branch-and-bound solver: no preprocessing; branching variable selection (*largest infeasibility*): take the variable with the largest integer infeasibility; node selection (*best bound*): in a minimization problem, take as the next node to process the one with the smallest value for the solution of its LP relaxation; no cutting-plane generation; no heuristic generation of integral solutions.

Using these settings, the problems *air01, air02, bm23, cracpb1, diamond, lp4l, misc01, misc02, misc04, mod013, p0040, pipex, sample2, sentoy, stein9,* and *stein15* all solved in 11 seconds or less and were selected for deletion. The solution of every other problem took at least 25 seconds. The model *air06* was also removed since it can be solved as a linear program, requiring no branching.

Various models were deleted to eliminate instances with very similar structures and similar computational properties:

bell3b, bell4 : The problems *bell3a, bell3b, bell4,* and *bell5* are all based upon the same underlying model. Both *bell3a* and *bell3b* are relatively easy; *bell4* and *bell5* seem about equally difficult, with *bell5* the more difficult of the two.

misc05 : The problems *misc05* and *misc06* have very similar solution characteristics, and the "*misc*" set seemed overrepresented.

p0291 : The "*p*" set also seemed overrepresented.

fixnet3, fixnet4 : The problems *fixnet3, fixnet4,* and *fixnet6* were all randomly generated and have very similar solution characteristics. From the three, we selected one to keep, more-or-less at random.

set1al, set1cl : Again, *set1al, set1cl,* and *set1ch* were all randomly generated and have similar solution characteristics. *set1ch* appears to be the most difficult and was retained.

2.4 Constraint Classification

One of the most important techniques to successfully solve mixed integer programs is reformulation. Reformulation techniques, such as preprocessing and cut generation, are incorporated in almost all state-of-the-art mixed integer optimizers. Many reformulation techniques are based on specific structures that may be embedded in the constraint matrix. For example, generation of lifted cover inequalities can only be done if the constraint matrix contains knapsack inequalities. Therefore, knowledge about the structure of the constraint matrix is useful in determining whether certain reformulation techniques can be applied. In view of the above, we have decided to provide some basic information about the constraint matrix of the problems in the MIPLIB. For each of the problems in the MIPLIB, information is given about the types of constraints that appear in the constraint matrix. In defining a constraint type we use the symbol x to denote binary variables and the symbol y to denote general integer and continuous variables. Each constraint type will be an equivalence class with respect to complementing binary variables, i.e., if a constraint with term $a_j x_j$ belongs to a given type, then the constraint with $a_j x_j$ replaced by $a_j(1 - x_j)$ also belongs to that class. Consequently, the most general constraint that can appear in a mixed integer program can be represented as follows

$$\sum_{j \in B} a_j x_j + \sum_{j \in I \cup C} a_j y_j \square b,$$

where B, I and C denote the sets of binary, integer, and continuous variables, respectively, a_j is positive for $j \in B$, a_j is nonzero for $j \in I \cup C$, and $\square$ is the sense of the constraint, either $\leq$ or $=$.

We distinguish the types of constraints given in Table 3.

Note that

$$\sum_{j \in N^+} x_j - \sum_{j \in N^-} x_j \leq 1 - |N^-|$$

with $N^+ \cup N^- = B$, is also considered to be a packing constraint, since complementing the variables with a negative coefficient gives

$$\sum_{j \in N^+} x_j + \sum_{j \in N^-} \bar{x}_j \geq 1$$

Table 3: Constraint types

G	General	$\sum_{j \in B} a_j x_j + \sum_{j \in I \cup C} a_j y_j \square b$
K	Knapsack	$\sum_{j \in B} a_j x_j \leq b$
E	Equality knapsack	$\sum_{j \in B} a_j x_j = b$
F	Facility location	$\sum_{j \in B} a_j x_j \square a_k x_k$
I	Invariant knapsack	$\sum_{j \in B} x_j \square k \; (k \geq 1 \wedge k \geq \lvert B \rvert - 1)$
P	Packing	$\sum_{j \in B} x_j \leq 1$
C	Covering	$\sum_{j \in B} x_j \geq 1$
S	Special ordered set	$\sum_{j \in B} x_j = 1$
U	variable Upper bound	$a_j y_j \square a_k x_k$
L	variable Lower bound	$a_j y_j \geq a_k x_k$

Table 4: Constraint types

NAME	G	K	E	F	I	P	C	S	U	L
10teams	110					40		80		
air03								124		
air04								823		
air05								426		
arki001	1027	13			6	2				
bell3a	101					22				
bell5	76					15				
blend2	93				79		9		88	
cap6000		7				2046		123		
dano3mip	2518								606	30
danoint	256								392	
dcmulti	165					22	10		45	45
dsbmip	1142							40		
egout	43								55	
enigma			1					20		
fast0507						3	504			
fiber	44							90		
fixnet6	100								378	
flugpl	18									
gen	318	24							432	6
gesa2	912					14			288	144
gesa2_o	624					48	144		288	144
gesa3	936					48			264	120
gesa3_o	672					48	120		264	120
gt2	27					2				
harp2		30		9				73		
khb05250	77								24	
l152lav		1						95		
lseu		11				17				
misc03	1	3		2	30	3	31	5		
misc06	820									
misc07	1	3		2	42	3	127	7		
mitre		383			1148		140	383		
mod008		6								
mod010		1						144		
mod011	4400							16	64	
modglob	95								196	
noswot	137								20	25
nw04								36		
p0033		11				4				
p0201		33			54	26	20			
p0282		64				177				
p0548		112				64				
p2756		403				352				
pk1	45									
pp08a	72								64	
pp08aCUTS	182								64	
qiu	664								528	
qnet1	178			32		4	4	48		45
qnet1_o	152			32				48		32
rentacar	6674								55	55
rgn	20					4				
rout	47					14			230	
set1ch	252								240	
seymour						285	4659			
stein27					1		117			
stein45					1	1	329			
vpm1	66								168	
vpm2	66								168	

Table 4 contains information regarding the different types of constraints occurring in each problem as well as the actual number of constraints of each type. For example, the row corresponding to problem *p0033* has entries 11 and 4 in columns K and S, respectively, indicating that eleven knapsack and four special ordered set constraints are present.

3 MIPLIB Availability

The MIPLIB library is accessible via the Internet. The MIPLIB page can be found at *http://www.caam.rice.edu/~bixby/miplib/miplib.html*.

The library can also be obtained via anonymous ftp. The library and related files are in the directory */pub/people/bixby/miplib* on the machine *ftp.caam.rice.edu*.

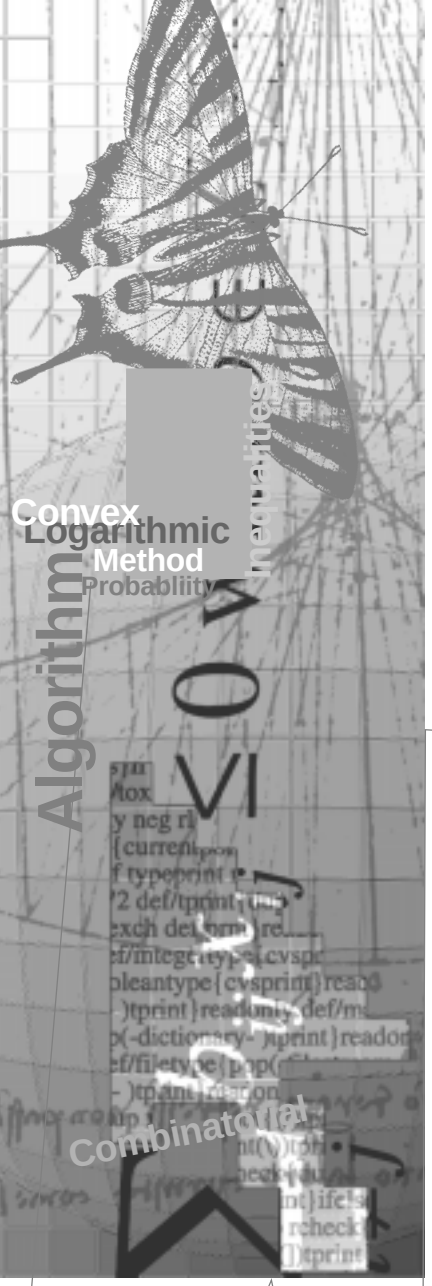
4 MIPLIB submissions

As noted earlier, we encourage both researchers and practitioners in integer programming to submit real-world instances to the authors for consideration and possible inclusion in MIPLIB. The library will be updated at least once each year after review of the submitted problems. In order to submit a problem, the MPS file containing the problem should be transferred into the directory *pub/people/bixby/miplib/incoming* on the machine *ftp.caam.rice.edu*. The submission should be accompanied by an e-mail message to *miplib@caam.rice.edu* containing information about the origin of the instance, the optimal solution (if known), and the way the optimal solution was obtained.

References

[1] Bixby, R.E., E.A. Boyd, and R.R. Indovina. 1992. MIPLIB: A Test Set of Mixed Integer Programming Problems. SIAM News 25:2 (March).

[1] CPLEX is a trademark of CPLEX Optimization, Inc.



Conference Notes

4th International Conference on Optimization
July 1-3, 1998, Perth, Australia
URL: http://www.cs.curtin.edu.au/maths/icota98

APPROX 98 - 1st International Workshop on Approximation Algorithms for Combinatorial Optimization Problems
July 18-19, Aalborg, Denmark
URL: http://www.mpi-sb.mpg.de/~approx98/

Optimization 98
July 20-22, 1998, Coimbra, Portugal
URL: http://www.it.uc.pt/~opti98

2nd WORKSHOP ON ALGORITHM ENGINEERING, W A E '98
August 19-21, 1998, Saarbruecken, Germany
URL: http://www.mpi-sb.mpg.de/~wae98/

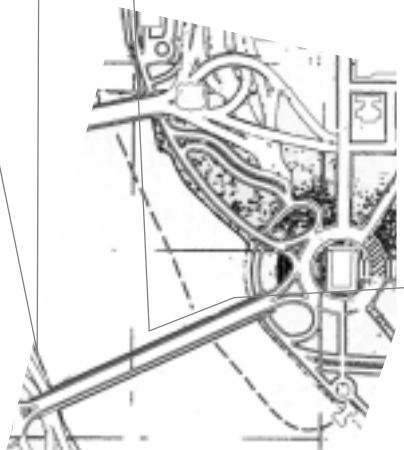
INFORMS National Meeting
October 25-28, 1998, Seattle WA
URL: http://www.math.org/seattleinforms.html

International Conference on Nonlinear Programming and Variational Inequalities
December 15-18, 1998, Hong Kong
URL: http://www.cityu.edu.hk/ma/conference/icnpvi/icnpvi.html

Sixth SIAM Conference on Optimization
May 10-12, 1999, Atlanta, GA

Fourth International Conference on Industrial and Applied Mathematics
July 5-9, 1999, Edinburgh, Scotland
URL: http://www.ma.hw.ac.uk/iciam99/

19th IFIP TC7 Conference on System Modelling and Optimization
July 12-16, 1999, Cambridge, England
URL: http://www.damtp.cam.ac.uk/user/na/tc7con



ICM'98

International Congress of Mathematicians

Berlin, Germany
August 18-27, 1998

Web Site

Information about ICM'98 can be found at the ICM'98 web site (http://elib.zib.de/ICM98).

Special Activities Related to Women in Mathematics

All who are interested in the participation of women in the study of mathematics and in the community of mathematicians are invited to the following activities:

FRIDAY, August 21, 19:30
PANEL DISCUSSION: After recognition of the involvement of women from many countries as ICM participants, women speakers from several countries will discuss "Events and policies: Effects on women in mathematics."
The panel is being organized by women from the Association for Women in Mathematics (AWM), the European Women in Mathematics (EWM) and the Committee on Women and Mathematics of the European Mathematical Society, represented by a committee consisting of Bhama Srinivasan (chair; Chicago, IL, USA), Bettye Anne Case (Tallahassee, FL, USA), and Christine Bessenrodt (Magdeburg, Germany). The organizers have received planning advice from women in several additional countries. They envision that each speaker will talk about how certain events or policies in her country have affected women in mathematics.
For more information, please contact Bettye Anne Case (case@math.fsu.edu).

FRIDAY, August 21, 21:15
A film entitled "Women and Mathematics across Cultures" will be shown. The film briefly introduces EWM, shows some statistics, and allows four female mathematicians to share their personal experiences about the impact of cultural differences on the status of women in the profession. The film was directed by Marjatta Naatanen (Helsinki, Finland) in collaboration with Bodil Branner (Lyngby, Denmark), Kari Hag (Trondheim, Norway), and Caroline Series (Warwick, UK).
For more information, please refer to the web page (http://www.math.helsinki.fi/EWM).
SATURDAY, August 22, 11:00
Cathleen Synge Morawetz, Courant Institute, New York University, will present an EMMY NOETHER LECTURE called "Variations on Conservation Laws for the Wave Equation." The Emmy Noether Lecture will be chaired by Irene M. Gamba (Austin, TX, USA).
ABSTRACT: The time dependent wave equation has many conservation laws obtainable by using Emmy Noether's theorem for equations coming from Lagrangians. From this nucleus we survey some estimates that can be found for equations close and not so close to the wave equation and show applications for these estimates (time decay for exterior problems and nonlinear Klein-Gordon, for the reduced wave equation and for the Tricomi equation). On a slightly different note, some weakly quasi and other nonlinear perturbations of the time wave equation have simple formal asymptotic solutions. These formal solutions probably represent real solutions but that requires some new estimates. For more information:, please contact Cathleen Synge Morawetz (morawetz@cims.nyu.edu).
– Bettye Anne Case (Tallahassee, Florida, USA), and Sylvia Wiegand (Lincoln, Nebraska, USA)

Berlin as a Centre of Mathematical Activity

Another event of general interest has reached its final shape. The workshop "Berlin as a Centre of Mathematical Activity," suggested by the International Commission on the History of Mathematics (ICHM) and organized on its behalf by G. Israel (Rome) and E. Knobloch (Berlin), was approved as one of the events of the "Section of Special Activities." It will take place on the afternoon of Saturday, August 22. The program of this workshop is as follows:
14.00 - 14.15 h G. Israel, University of Rome: *Introduction*
14.15 - 14.45 h David Rowe, University of Mainz: *The Berlin-Goettingen Rivalry, 1870-1920*
14.45 - 15.00 h Discussion
15.00 - 15.30 h Ivor Grattan-Guinness, Middlesex Polytechnic, England: *Weierstrassian analysis and Cantorian set theory in Britain and the USA, 1890-1910*
15.30 - 15.45 h Discussion
15.45 - 16.15 h Coffee break
16.15 - 16.45 h U. Bottazzini, Centro Linceo Interdisciplinare 'B. Segre' Accademia dei Lincei, Rome: *Weierstrass's school of analysis and its influence on Italian mathematics*
16.45 - 16.50 h Discussion
16.50 - 17.20 h S. Demidov, Russian Academy of Sciences, Moscow: *Russian mathematicians in Berlin in the 2nd half of the XIXth and at the beginning of the XXth centuries*
17.20 - 17.25 h Discussion
17.25 - 17.55 h A. Dahan, CNRS Paris: *Images croisées des centres mathématiques de Paris et de Berlin, dans les années 1860/80*
17.55 - 18.00 h Discussion

URANIA Activities

At its last meeting in August 1994 at Luzern, Switzerland, the IMU General Assembly (GA) approved the following:
RESOLUTION 5: "The General Assembly recommends that the Programme Committee schedule some less formal scientific events of broad interest during the Congress."
The ICM'98 Organizers agree that it is of utmost importance to explain general ideas of our science and to indicate research trends to the educated layman and to make connections of mathematics to other areas of science, technology, economics, music, art, philosophy, etc., visible.
To achieve some of these goals, the ICM'98 Organizing Committee has rented the Urania building to present mathematics to a broad audience from August 20 to 27, 1998. The Urania building belongs to the Urania foundation, a well established organization founded by important industrialists and famous scientists more than 100 years ago in order to popularize scientific research (in a broad sense). The man-

►

agement of Urania will support the advertisement of the mathematical events through its well-established channels.
The program to be offered at the Urania is currently taking shape. The activities are organized by Ehrhard Behrends (FU Berlin, behrends@math.fu-berlin.de). You will obtain the final program with your conference material; updates can, of course, be found at the ICM'98 web site (http://elib.zib.de/ICM98/C/1/urania). Although the Urania program is directed to the general public, we believe that it will also be of interest for ICM'98 participants and for accompanying persons, in particular.

Footloose Tours, A First Glimpse

To convey some of the many facets of Berlin to the ICM'98 participants and to accompanying persons, especially, several Berlin mathematicians, their friends and spouses have volunteered to offer informal tours of Berlin, especially to places "off the beaten track." Most of the tours are free of charge, except for possible entrance fees (and public transportation fares for those accompanying persons who have not bought the transportation ticket offered by ICM'98). In a few cases professional guides with particular expertise have been hired. The list below is preliminary and is just intended to give you an idea of what the Berlin mathematicians are planning as a particular service for their ICM'98 guests. The final list of footloose tours will be distributed via the ICM'98 E-mail information service, probably in early July.

We want to keep the organization of the footloose tours very informal. Booking will be possible only via e-mail. (Special e-mail addresses for footloose tour registration will be announced with the final list of these tours.) Places will be assigned on a first-come, first-serve basis. There will, however, be a booth at the Congress registration desk to help those who could not book through e-mail, to handle last minute changes and, if possible, organize additional ad-hoc tours or the repetition of tours for which there was particularly great interest. We want to keep groups small. Thus, group sizes are tightly restricted. One exception is the organ recital in the Berliner Dom where sufficiently many seats are available and for which you can already register.
Unless stated otherwise, the tours will start in front of the main building of the Technical University, the site of the Congress. Ralph-Hardo Schulz (FU Berlin) is in charge of the organization of the footloose tours. If you have any questions or suggestions please contact him via e-mail (schulz@math.fu-berlin.de).

Preliminary List of Footloose Tours

1. Berliner Dom - Guided Visit and Organ Recital; 2. Guided Tours of Berlin in Russian; 3. Visite de Berlin Guide en Francais; 4. Visita Guidata di Berlino in Italiano; 5. Tour through the New Berlin (Guided Tour in English); 6. Berlin Alternative 7; Path to the Jewish Berlin; 8. "Fascism: Seizing of a town"; 9. Gedenkstaette deutscher Widerstand; 10. Walk along the Former Border Line (Wall); 11. A Visit to Humboldt University; 12. Tombs of Weierstrass and Kronecker; 13. Cemetery Tour; 14. Gipsformerei (Plaster Moulders of the Berlin Museums); 15. Egyptian Museum and Park of Charlottenburg Castle; 16. Gemaeldegalerie Kulturforum (Picture Gallery); 17. Picasso (the Berggruen collection); 18. Deutsche Bank (Guggenheim Museum); 19. KPM Werkstatt ("Royal Prussian Porcelain Manufacture"); 20. Guided Walk through the Zoo; 21. Riksha Tour from Alexanderplatz to the Brandenburg Gate; 22. Visit of "Hackesche Hoefe"; 23. Bus No. 100 from Bahnhof Zoo to Alexanderplatz and Back; 24. Strolling through the Botanical Garden; 25. Botanical Garden (Guided Tour); 26. Shopping in KaDeWe; 27. Walk through the Park of Schloss Glienicke; 28. Walk through the Grunewald; 29. Walk around the Straussee (a lake near Berlin).
The ICM'98 web site (http://elib.zib.de/ICM98/E/5) now includes a page about the ICM'98 footloose tours. More details will be added in the near future about the tours that will be offered.
–Written on behalf of Martin Groetschel, President of the ICM'98 Organizing Committee

International Conference on Nonlinear Programming and Variational Inequalities

We are pleased to announce that the International Conference on Nonlinear Programming and Variational Inequalities will be held December 15-18, 1998, in Hong Kong. The conference aims to review and discuss recent advances and promising research trends in some areas of Nonlinear Programming and Variational Inequalities. Since information on the conference was released in July, there has been a very good response from the mathematical community. Around 200 people have already shown interest in attending. You are welcome to join with us as well.
Invited speakers of the conference include professors from universities in the USA, Canada, Australia, Japan and Europe. Key speakers include Professor M. Powell (University of Cambridge, UK), Professor R.J.B. Wets (University of California at Davis, USA), Professor M. Kojima (Tokyo Institute of Technology, Japan), Professor J.S. Pang (Johns Hopkins University, USA), and Professor Y. Yuan (Chinese Academy of Sciences, China).
Two international mathematics journals have recognized the importance of this conference by agreeing to publish special issues commemorating the conference. These are the Journal of Computational and Applied Mathematics, and the Journal of Optimization Methods and Software.
If you are interested in joining this conference, please see web site http:\\www.cityu.edu.hk\ma\conference\icnpvi for details, or contact me via e-mail (maopt@cityu.edu.hk) for more information.
– PEGGY CHAN,
ICNPVI CONFERENCE SECRETARY

Reviews

Interior Point Methods of Mathematical Programming

Edited by T. Terlaky
Kluwer Academic Publishers
ISBN 0-7923-4201-1

This book consists of a collection of articles on interior-point methods, written by leading experts in the field. Each article forms a chapter of the book. The editor was careful to solicit a nice collection of articles covering the entire field of interior-point methods with very little, if any, unnecessary overlap. Below I briefly describe what is covered in each of the chapters.

Part I: Linear Programming

Chapter 1. Introduction to the Theory of Interior Point Methods (B. Jansen, C. Roos, and T. Terlaky)
This chapter lays the groundwork for the rest of the book. In it, the authors prove the strong duality theorem and the strict complementarity theorem using logarithmic barrier functions for the skew-symmetric self-dual problem. It also includes a detailed discussion of sensitivity analysis purely from the perspective of interior point methods.

Chapter 2. Affine Scaling Algorithms (T. Tsuchiya)
This chapter provides an extensive survey of results on affine scaling methods starting with the historical significance of these methods and continuing with a detailed analysis of their convergence properties all the way up to the surprising result that the primal variables and dual estimates converge when and only when the step length is at most $^2/_3$ of the way to the boundary of the feasible region. Also presented is Mascarenhas' example showing that the primal variables can converge to a nonoptimal vertex when problems are degenerate and step lengths are very close to one.

Chapter 3. Target Following Methods for Linear Programming (B. Jansen, C. Roos, and T. Terlaky)
This chapter introduces an approach to interior-point methods based on the *space of complementary products*, or *v*-space. This approach can be applied in many variations and yields some of the simplest and most elegant complexity proofs.

Chapter 4. Potential Reduction Algorithms (K. Anstreicher)
Karmarkar's algorithm, which inspired the whole subject of interior-point methods, had three novel aspects: (i) a nonstandard formulation for the linear programming problem, (ii) projective transformations, and (iii) a potential function to measure the progress of the algorithm.

It was quickly realized that similar results could be obtained with algorithms that worked on standard forms and without projective transformations. However, the projective transformation seemed fundamental for quite some time and many variants and convergence results were obtained for so-called potential reduction algorithms. This chapter surveys these algorithms and their associated complexities.

Chapter 5. Infeasible Interior Point Algorithms (S. Mizuno)

In this chapter, the so-called infeasible interior-point methods are discussed and analyzed. In contrast to the earlier chapters, this chapter focuses on algorithms that are directly related to those methods that (currently) appear to be the most efficient in practice. These methods are one-phase methods in which each iteration attempts to decrease three things simultaneously: primal infeasibility, dual infeasibility, and duality gap. The chapter primarily considers two variants: one that is close to what people implement in practice and another for which good convergence and complexity results exist.

Chapter 6. Implementation Issues (E. Andersen, J. Gondzio, C. Mészáros, and X. Xu)

Building on the shift toward practical implementations started by the previous chapter, this chapter discusses many important specific implementation details such as preprocessing, sparse linear algebra, adaptive higher-order methods, initialization, and stopping strategies. Also discussed are the effect of centering, cross-over to a basis, and basis identification.

Part II: Convex Programming

Chapter 7. Interior Point Methods for Classes of Convex Programs (F. Jarre)

Much of this chapter is an introduction to self-concordant barrier functions introduced by Nesterov and Nemirovsky as a means of generalizing interior-point methods to the broader domain of smooth convex optimization problems. Jarre has done an excellent job of distilling this important subject into a very understandable form.

Chapter 8. Complementarity Problems (A. Yoshise)

This chapter covers the extension of interior-point methods to linear and nonlinear complementarity problems. This class of problems is a very natural extension of linear programming and includes several important real-world problems such as equilibrium problems and variational inequalities. Much of the chapter deals with complexity results for the monotone complementarity problem.

Chapter 9. Semidefinite Programming (M. Ramana, P. Pardalos)

Semidefinite programming is a generalization of linear programming in which the nonnegative vector of decision variables is replaced by a square positive semidefinite matrix of such variables. This class of problems has been of particular interest lately since there are important applications to engineering design problems and to combinatorial optimization problems and because Alizadeh discovered that interior-point methods provide a very natural way to solve these problems. This chapter surveys the problem domain, the application areas, the interior-point algorithms, and the associated duality theory. There is a particular emphasis on this latter topic.

Chapter 10. Implementing Barrier Methods for Nonlinear Programming (D. Shanno, M. Breitfeld, and E. Simantiraki)

This chapter revisits the classical logarithmic barrier method introduced by Fiacco and McCormick with the added insight provided by the last 10 years of interior-point method development. Algorithms are presented for nonconvex nonlinear programming problems and preliminary results using the Hock and Schittkowski test suite are provided.

Part III: Applications and Extensions

Chapter 11. Interior Point Methods for Combinatorial Optimization (J. Mitchell)

This chapter surveys work on applying interior-point methods to the solution of combinatorial optimization problems. Included are discussions of branch and cut methods for integer programming, a potential reduction method based on transforming an integer programming problem into an equivalent nonconvex quadratic programming problem, interior-point methods for network flow problems, and methods for solving multicommodity flow problems, including an interior-point column-generation algorithm.

Chapter 12. Interior Point Methods for Global Optimization (P. Pardalos and M. Resende)

The application of interior-point methods to global optimization problems is still a subject in its infancy. Nonetheless, promising research directions have emerged and this chapter surveys some of them. As in the case of combinatorial optimization, most problems in global optimization are *NP*-hard. Hence, most work involves finding good approximate solutions.

This chapter describes some potential reduction and affine-scaling algorithms for nonconvex quadratic programming problems and for some classes of combinatorial optimization problems. It also provides some new lower bounding techniques for these problems.

Chapter 13. Interior Point Methods for the VLSI Placement Problem (A. Vannelli, A. Kennings, and P. Chin)

This final chapter discusses the application of interior-point methods to certain specific optimization problems arising in VLSI chip design. There are many, many real-world application areas for optimization technology. It is therefore a bit strange that this book only considers one such area. Nonetheless, VLSI chip design is an interesting application and readers will probably enjoy reading about it.

Conclusion

This book is an excellent reference work on all aspects of interior point methods. Unfortunately, its price is prohibitive – but do look for it in your local library.

– ROBERT VANDERBEI

Linear programming: foundations and extensions

by Robert J. Vanderbei

Kluwer Academic Publishers

ISBN 0-7923-9804-1

[Due to the change of editorial board, reviews of Vanderbei's book were acquired independently by the outgoing and incoming book review editors. We have decided to publish them both. For the first review, see pg 12, OPTIMA No. 56.–Editor]

During the past decade there has been a lot of activity in the area of linear programming mainly due to the introduction of the polynomial time interior-point methods. This seems to have inspired Robert J. Vanderbei to write a new book with the title "Linear programming: Foundations and extensions" which gives a lively and elementary introduction to both classical as well as more recent topics in LP.

The book consists of four parts discussing respectively the simplex method and duality, network models, interior-point methods, and extensions to non-linear problems.

Quite naturally in Chapter 1, the book begins by showing an example of an LP model to motivate the use of LP. Next, in Chapter 2, the simplex method is derived using a dictionary approach and not the classical simplex tableau. This has the advantage that the underlying ideas of the simplex method is better exposed to the reader. Hence, the author's presentation is in many ways similar to that of Balinski & Tucker (1969), and Chvátal (1983). In the following chapters degeneracy and efficiency issues are discussed. In Chapter 5, LP duality theory is presented. This chapter has two very nice features: first, it emphasize how duality can be used to establish bounds on the optimal objective function value; second, the dual simplex method is derived in a very natural way. In Chapter 6 and 7 the matrix based simplex method and sensitivity analysis is presented. The chapter on sensitivity analysis is too brief and does not discuss some of the pitfalls using the traditional sensitivity analysis [ch. 19] (Roos, Terlaky and Vial, 1997). Chapter 7 also includes a presentation of the self-dual simplex algorithm which seems to be the simplex method preferred by the author. Implementation issues are discussed in Chapter 8 and 9. In particular, the topics sparsity, the LU factorization of the basis and its update, pricing strategies, and general bounds on the variables are treated.

This discussion could be improved by treating partial pivoting which is necessary for numerical stability. Also a discussion of the real Markowitz pivoting strategy or at least a citation of the paper by Suhl & Suhl (1990) would have been appropriate. At the end of part one of the book there are three chapters about how LP duality can be used in convex analysis to prove Farkas' lemma and the existence of a strictly complementary solution for instance. In addition, the use of LP in game theory and regression analysis is covered.

The second and brief part of the book presents network-type models. Basically, it is observed that there is a one-to-one correspondence between spanning trees and bases for network flow problems. Moreover, if this fact is exploited, then all the variants of the simplex algorithm can be executed much faster. This part also has a chapter about structural optimization using network models.

The third part of the book is about interior-point methods, which are introduced using the logarithmic barrier method. This gives an easy and general introduction to the basic ideas of interior-point methods. The important concept of the central path is introduced and the so-called infeasible path-following primal-dual algorithm is derived. In Chapters 18 and 19, issues related to implementing interior-point methods are discussed. Especially the subject of efficient and numerically stable computation of the search direction is treated whereas issues such as using a different step size in the primal and dual space, and Mehrotra's predictor-corrector method is not covered. The last two chapters of part three contain a discussion on affine scaling methods and the homogeneous self-dual interior-point method. The presentation of the homogeneous self-dual method includes a proof of polynomial complexity using a Mizuno-Todd-Ye type predictor-corrector algorithm. This important chapter is unfortunately not as well written as most of the other chapters of the book. The homogeneous model and its properties could have been better separated from the presentation of the homogeneous algorithm leading to a better emphasis of the infeasibility detection capability of the homogeneous model.

The fourth and last part of the book contains three chapters about integer programming, convex quadratic programming (QP), and general convex programming. The chapter about integer programming explains the branch and bound method in some detail. The main features of the QP chapter is a presentation of the Markowitz portfolio choice model and a generalization of the primal-dual interior-point method to convex QP. The chapter about convex programming is very brief, but manages to present a successive quadratic programming (SQP) type method for solution of convex optimization problems where the subproblems of course are solved using an interior-point algorithm. However, the important topics of doing a line search and convergence are not treated.

Finally, it should be mentioned that the author maintains a web site which includes additional material related to the book. This includes C source code for a number of the algorithms presented from the book, an interactive Java based pivot tool, and additional exercises.

Overall, Vanderbei's book gives a modern, comprehensive, and well-balanced introduction to LP and related topics. The material in the book is presented at an elementary level which makes it easy for students as well as newcomers to the area to read and understand the book. It has many nice numerical examples, which will definitely be appreciated. Moreover, the book prepares the reader well for further studies of LP.

I also have a few reservations with respect to the book. The notation is occasionally more complicated than necessary and is not always standard. Moreover, I would have preferred if the author had used equalities in his standard LP form because this reduces the number of symbols to keep track of. The exercise sections of the book could be improved and extended. In many cases the author prefers using an example instead of a real proof, which may not always be ideal. A few subjects are absent in the book; in particular, decomposition methods and semi-definite programming are not mentioned at all.

In conclusion, Vanderbei's book gives an excellent introduction to linear programming, especially the algorithmic side of the subject. The book is highly recommended for both self study and as teaching material.

References

M. L. Balinski and A. W. Tucker. Duality theory of linear programs: a constructive approach. *SIAM Rev.*, 11(3):347-377, July 1969.

V. Chvátal. *Linear programming*. W.H. Freeman and Company, 1983.

C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and algorithms for linear optimization: an interior point approach*. John Wiley & Sons, New York, 1997.

U. H. Suhl and L. M. Suhl. Computing sparse LU factorizations for large-scale linear programming bases. *ORSA J. on Comput.*, 2(4):325-335, 1990.

–ERLING D. ANDERSEN

►

Annotated Bibliographies in Combinatorial Optimization

Edited by M. Dell' Amico, F. Maffioli and S. Martello
John Wiley & Sons (1997)

ISBN 0-471-96574-X

Summary

This book presents a quite thorough and up-to-date overview of the field of Combinatorial Optimization, with a detailed analysis of the state-of-the-art in many relevant areas of study and, in particular, annotated bibliographies thereof.

Recognized research leaders examine individual topics: Hardness of Approximation, Polyhedral Combinatorics, Branch-and-Cut Algorithms, Matroids and Submodular Functions, Perfect, Ideal and Balanced Matrices, Advances in Linear Optimization, Decomposition and Column Generation, Stochastic Integer Programming, Randomized Algorithms, Local Search, Sequencing and Scheduling, The Traveling Salesman Problem, Vehicle Routing, Max-Cut Problem, Location Problems, Flows and Paths, Network Design, Network Connectivity, Linear Assignments, Quadratic and Three-Dimensional Assignments, Cutting and Packing, Set Covering Problem, Combinatorial Topics in VLSI Design, and Computational Molecular Biology. There is also a chapter on books in Combinatorial Optimization.

In my opinion this book will be of very high value to researchers, educators and even practitioners. The organization of the book and the lightweight style in which the different chapters are (mostly) written makes it easy for somebody to quickly obtain a snapshot of the state and direction of a particular field and relevant references.

Why you need this book.

Combinatorial Optimization, as a look at the above list of topics shows, has grown explosively in recent years. This growth has been fueled by areas of application that yield difficult problems, by major advances in the foundations of computer science and discrete optimization, and by a renewed interest in effective computation. It is doubtful that anybody could claim to be familiar with every single area of recent activity.

This book makes it easy to become an instant expert on any of the above fields — after a suitable amount of reading of references, of course. The book will also be useful to an expert in a particular field, by gathering together and annotating references. I have already used the book to locate articles where particular results are described (I knew the result, I knew the author, and I even knew the journal, but not the particular reference).

The structure of the book.

Each area of study is analyzed either historically or by sub-fields. An overview of the current state of the field is presented together with the references. Usually, a road map to some of the results, and even a sketch of some crucial proofs is also given. I found this feature particularly enjoyable.

I was also quite pleased to see the chapters on Hardness of Approximation (V. Kann and A. Panconesi) and Randomized Algorithms (M. Goemans, D. Karger and J. Kleinberg) included in this volume. These chapters discuss recent results and techniques that will have deep and lasting impact in our field, and yet may not be completely familiar to many researchers in Combinatorial Optimization. The chapter on Branch-and-Cut Algorithms (A. Caprara and M. Fischetti) details some emerging techniques in the successful solution of real-world integer programs. Some exciting new results, possibly (eventually) leading to a proof of the Strong Perfect Graph Conjecture, are presented in the chapter on Perfect, Ideal and Balanced Matrices (M. Conforti, G. Cornuéjols, A Kapoor, K. Vuskcovic). The Set Covering Problem chapter (S. Ceria, P. Nobili and A. Sassano) presents a historical overview and also describes some promising new techniques in the solution of this very important class of problems. The chapters on Vehicle Routing (G. Laporte) and Location Problems (M. Labbé and F.V. Louveaux) are a testament to the increasing importance of these logistical problems. Other important applications with substantial combinatorial content are described in the chapters on Network Design (A. Balakrishnan, T.L. Magnanti and P. Mirchandani), Combinatorial Topics in VLSI Design (R. Möhring, D. Wagner) and Computational Molecular Biology (M. Vingron, H.-P. Lenhof and P. Mutzel). This partial list reflects my personal interests, but in fact all chapters are useful, substantial and generally well written.

Critique.

I found this book quite satisfactory, and can only point out some minor issues.

Even though the book is fairly thorough, perhaps it could have included chapters on Computational Geometry and on the Geometry of Numbers. These are substantial topics which arguably would have fit well. Also, even though computational aspects of integer programming are frequently addressed (most notably, in the branch-and-cut chapter) perhaps a unified treatment could have been desirable — although I realize that making such a chapter fit without too much overlap with others might not be an easy task.

On a different note, the focus of this book is, of course, the references provided therein. Many readers are likely to want to read some of these references, in particular the more recent ones, after finding them here. Nowadays, a common way of obtaining articles is through the Web. Ideally, the book could have included URLs for the download of at least some papers — although I hasten to add that the creation and maintenance of such a database of URLs would be a very difficult logistical undertaking, possibly beyond the scope of a book.

Finally, overall the references appear complete. I found only a few cases of references that I would have included; and perhaps even some questionable references. But this may largely be a matter of taste, and in any case a reader interested in a particular topic is likely to find any "missing" references while reading those provided here!

In conclusion: this is an excellent book, a "must have" for anybody with an interest in Combinatorial Optimization.

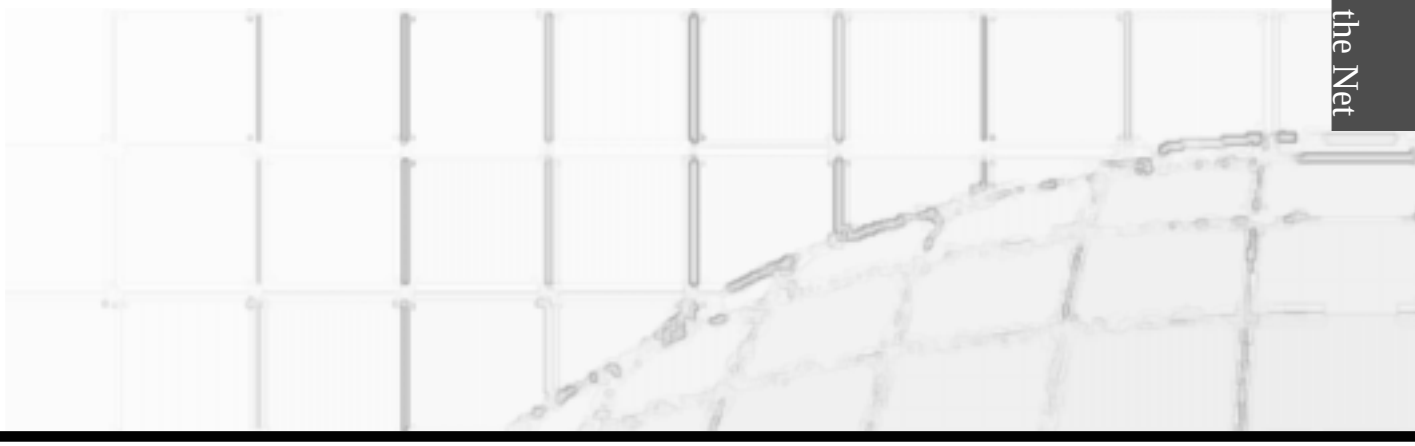
–DAN BIENSTOCK

Rainer Burkard has been elected Honorary Member of the Hungarian Academy of Sciences.

Deadline for the next O P T I M A September 15, 1998

For the electronic version of OPTIMA, *please see*:

http://www.ise.ufl.edu/~optima/



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 NUMBER: _____ EXPIRY DATE: _____

FAMILY NAME: _____

MAILING ADDRESS: _____

_____

TEL.NO.: _____ TELEFAX: _____

E-MAIL: _____

SIGNATURE _____

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

Checks or money orders should be made payable to The Mathematical Programming Society, Inc. Dues for 1998, including subscription to the journal *Mathematical Programming*, are US $70. 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

_____

O P T I M A
MATHEMATICAL PROGRAMMING SOCIETY

UNIVERSITY OF
FLORIDA

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

FIRST CLASS MAIL

published by the
MATHEMATICAL PROGRAMMING SOCIETY &
GATOREngineering® PUBLICATION SERVICES
UNIVERSITY OF FLORIDA