Approximation Algorithms*

Andreas S. Schulz[1] David B. Shmoys[2] David P. Williamson[3]

[1] Fachbereich Mathematik, Technische Universität Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany
[2] School of Operations Research and Industrial Engineering and Department of Computer Science, Cornell University, 232 Rhodes Hall, Ithaca, NY 14853
[3] IBM T. J. Watson Research Labs, P. O. Box 218, Yorktown Heights, NY 10598

Abstract

Increasing global competition, rapidly changing markets, and greater consumer awareness have altered the way in which corporations do business. To become more efficient, many industries have sought to model some operational aspects by gigantic optimization problems. It is not atypical to encounter models that capture 10^6 separate "yes" or "no" decisions to be made. Although one could, in principle, try all 2^{10^6} possible solutions to find the *optimal* one, such a method would be impractically slow. Unfortunately, for most of these models, no algorithms are known that find optimal solutions with reasonable computation times. Typically, industry must rely on solutions of unguaranteed quality that are constructed in an ad hoc manner. Fortunately, for some of these models there are good *approximation algorithms*: algorithms that produce solutions quickly that are provably close to optimal. Over the past six years, there has been a sequence of major breakthroughs in our understanding of the design of approximation algorithms and of limits to obtaining such performance guarantees: this area has been one of the most flourishing areas of discrete mathematics and theoretical computer science.

*This paper is a summary of a session presented at the third annual German-American Frontiers of Science symposium held June 19-22, 1997, at the Kardinal Wendel Haus, Munich, Germany. It is to appear in the Proceedings of the National Academy of Sciences of the USA.

1 Introduction

Many optimization problems are believed to be intractable computational problems; that is, there is strong mathematical evidence to support the hypothesis that there do not exist algorithms guaranteed to find optimal solutions quickly. This prompted the study of approximation algorithms, in which the aim is to find provably near-optimal solutions quickly. In the past few years, there have been major advances in the design and analysis of approximation algorithms; we briefly will outline some of these algorithmic techniques.

Before defining the mathematical formalisms that correspond to the intuitive phrase "computing good solutions to hard discrete optimization problems quickly," we first motivate this with an example. In the economical manufacturing of printed circuit boards, the following problem arises: holes have to be drilled through the board at given positions (see Figure 1), and we want to compute the order in which to drill the holes so as to minimize the total time spent moving the head of the drill. This problem has been the subject of much research, although under its usual name, the traveling salesman problem. It is a *discrete* optimization problem; there is a finite number of possible solutions (called tours), and we could find the shortest tour, in principle, by trying them all.

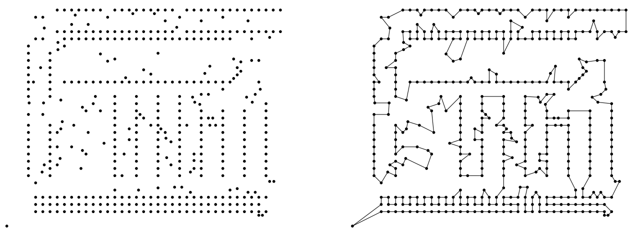

Figure 1: An input to the drilling problem and a corresponding tour.

However, if there are n holes, then there are $(n-1)!/2$ different tours. For example, for $n = 50$, even under wildly optimistic assumptions about the speed of super-computers, it would take more than 10^{39} years to find the optimal solution by such exhaustive search! The running time of this algorithm grows as an exponential function of the size of the input, and in contrast, an algorithm is considered *efficient* if its running time can be bounded by a polynomial function of the input size (i.e., $3n$ or n^3, rather than 2^n or $(n-1)!/2$).

Unfortunately, most real-world optimization problems seem too hard to be solved efficiently. That is, no algorithm is known that is guaranteed to find an optimal solution efficiently, and in fact, even the most sophisticated methods used today fail to find optimal solutions for most large-scale applications. Even problems that sound simple, such as the drilling problem given above, are believed to be hard. Computational complexity theory and its notion of NP-completeness provide a mathematical foundation for this belief. NP is a rich class of problems, containing variants of virtually every optimization problem. NP-complete problems are the hardest problems in NP, in that, an efficient algorithm to solve any NP-complete problem, such as finding the best drilling pattern, also yields an efficient algorithm for every problem in NP. It is now widely accepted that NP-complete problems cannot be solved efficiently, but to prove this, i.e., to prove that $P \neq NP$, remains one of the most challenging open problems in mathematics.

However, these optimization problems still need to be solved in practice, and so we must settle for less. This leads to the concept of an approximation algorithm; an *α-approximation algorithm* must efficiently compute a solution of value within a factor of α of optimal. Thus, for any given problem, we wish to determine the smallest α for which we can find an α-approximation algorithm. There have been significant recent breakthroughs both in giving improved performance guarantees, and in proving limits on the extent to which near-optimal solutions can be efficiently computed. While we shall highlight only a few ideas in the former category and refer the reader to [1] for the latter, progress on both sides has made this one of the most flourishing areas of discrete mathematics and theoretical computer science.

2 Some Examples

The central difficulty in designing approximation algorithms is proving that a solution close to the optimal can be computed quickly, when the optimal solution itself cannot be computed quickly. To illustrate some of the recent techniques used in addressing this problem, we will focus first on the *maximum cut* problem. In this problem we are given n items (typically called *nodes*) which are numbered 1 through n, and pairs of items (i, j) (called *edges*) with associated weights $w_{ij} \geqslant 0$. The goal is to divide the set of nodes into two parts so as to maximize the sum of the weights of those edges whose nodes are in different parts. These edges are said to be *in the cut*. This NP-complete problem arises in various contexts, from finding the ground state of the Ising spin glass model in statistical physics to minimizing the number of holes that must be drilled in circuit boards; see Barahona et al. [2] for further details.

Randomization has proven to be a particularly effective tool in the design and analysis of approximation algorithms, and throughout discrete mathematics (see, e.g., [3, 4]). A naive use of randomization is to pick a solution uniformly at random; for example, in the maximum cut problem, we can flip a coin for each node, and thereby split the nodes into the "heads" set and the "tails" set. For each edge (i, j), the probability that (i, j) is in the cut of this random solution is exactly 1/2. Thus the expected weight of the random solution is $\frac{1}{2}\sum_{(i,j)} w_{ij}$. Since the total weight of all edges $\sum_{(i,j)} w_{ij}$ is clearly an upper bound on the value of an optimal solution, this proves that the expected weight of the random solution is within a factor of 2 of the value of an optimal solution. In fact, by considering the nodes in order and choosing the outcome of the coin for which the remaining conditional expectation is larger, we can *derandomize* this algorithm to yield a 2-approximation algorithm.

In order to produce better quality solutions, we perform some computation that will allow us to bias our random solution in a favorable way. Suppose we introduce a variable x_i for each node i. We wish $x_i = -1$ when i is in one set of an optimal solution, and $x_i = 1$ when i is in the other set. Hence, $\frac{1}{2}w_{ij}(1 - x_i x_j) = w_{ij}$ exactly if edge (i, j) is in the cut, and is 0 otherwise. Thus if we could efficiently find values $x_i \in \{-1, 1\}$ that maximize $\frac{1}{2}\sum_{(i,j)} w_{ij}(1 - x_i x_j)$, we would be able to solve the maximum cut problem. We do not know how to do this, but we can efficiently solve the following vector problem: we can maximize $\frac{1}{2}\sum_{(i,j)} w_{ij}(1 - v_i \cdot v_j)$ for unit-length vectors v_i in n-dimensional space, where $v_i \cdot v_j$ is the *inner product* of vectors v_i and v_j. This problem can be solved using *semidefinite programming*. Notice that this vector problem is a *relaxation* of the previous one: that is, for each solution with $x_i \in \{-1, 1\}$, we can construct a set of unit-length vectors v_i such that $\frac{1}{2}\sum_{(i,j)} w_{ij}(1 - x_i x_j) = \frac{1}{2}\sum_{(i,j)} w_{ij}(1 - v_i \cdot v_j)$. Thus the value W^* of an optimal solution to the vector problem is at least the value of an optimal solution to the maximum cut problem. If we can show that the solution to the vector problem can be used to construct a cut of weight not much less than W^*, then the cut obtained is provably near-optimal.

We now use the optimal solution to the vector problem to produce a solution to the maximum cut problem: we select a vector r at random, put node i in one set if $v_i \cdot r \geqslant 0$ and put node i in the other set if not. It is then possible to prove that the expected weight of the cut produced in this way is at least $0.878W^*$, proving that the expected weight is at least .878 of an optimal cut, or within 14% of optimal. Thus using the vector problem to bias our random choice of a solution helps us to produce significantly better solutions to the maximum cut problem. This use of semidefinite programming was introduced by Goemans and Williamson [5], and has subsequently been adapted to several other settings [6]. This result gave the first improvement in approximating the maximum cut problem after almost 20 years of essentially no progress.

Linear programming is the technique most frequently used to obtain strong performance guarantees. We shall illustrate this approach by the following problem of routing in a communication network, which consists of communication links connected at switching points. We are also given k requests, pairs of switching points (s_i,t_i), $i = 1,\dots,k$, which correspond to pairs of users that wish to communicate over this network. For each pair i, we need to choose one path from s_i to t_i in the network. The aim is to choose the paths so that the *congestion*, the maximum number of paths requiring the same link, is minimized.

We can formulate this problem as follows: let $\mathcal{P}_i$ denote the set of paths from s_i to t_i in the network, and for each path $P \in \mathcal{P}_i$ we use a variable x_P, which is set to 1 to denote that we use path P for the request (s_i,t_i), and is 0 otherwise. We must choose exactly one path for each request: this can be expressed with these variables, by requiring that each variable be either 0 or 1 and $\sum_{P\in\mathcal{P}_i} x_P = 1$, for each $i = 1,\dots,k$. If the congestion is C, then for each link ℓ in the network, we select at most C paths that contain ℓ; thus, the sum of all variables corresponding to these paths is at most C. Our aim is to minimize C subject to these simple constraints on the variables. This is called an *integer programming* formulation of the problem. If we relax the requirement that each variable be 0 or 1, and merely require that each variable be a value in the interval [0,1], then this is a *linear program* (LP). Integer programming, even of this special form, is an NP-complete problem, but LPs can be solved efficiently.

The optimal value C^* of this LP is a lower bound on the optimal congestion. If we can show that the optimal LP solution can be rounded to an integer solution of not much greater congestion, then the rounded solution is provably near-optimal. Raghavan & Thompson [7] introduced an elegant *randomized rounding* technique: interpret the value of $x_P \in [0,1]$ in the optimal LP solution as a probability, and for each $i = 1,\dots,k$, choose a path $P \in \mathcal{P}_i$ with probability x_P. For any link ℓ, it is easy to see that the expected number of paths selected that contain ℓ is at most C^*. Furthermore, with some additional technical conditions, it is not hard to argue that the probability is quite small that significantly more paths use ℓ, e.g., more than $(1+\varepsilon)C^*$ for any constant $\varepsilon > 0$. In fact, this probability for one link is sufficiently small that it is also likely that the congestion is at most $(1+\varepsilon)C^*$, and hence within a factor of $(1+\varepsilon)$ of the optimal congestion.

Finally, there also has been a dramatic recent advance for the drilling problem discussed in the introduction. If the time to move the drill is the Euclidean distance between the holes, Arora [8] and Mitchell [9] gave a $(1+\varepsilon)$-approximation algorithm, for any constant $\varepsilon > 0$.

These examples highlight the importance of strong relaxations in the design of approximation algorithms, and show the power of randomization in constructing good solutions. Other related important algorithmic techniques also have contributed to surprising advances in this area, and the reader is referred to [10, 11] for more comprehensive surveys of approximation algorithms.

Acknowledgments

David Shmoys has been supported in part by NSF grant CCR-97-00029.

References

[1] Arora, S. & Lund, C. (1997) in [11], pp. 399–446.

[2] Barahona, F., Grötschel, M., Jünger, M., & Reinelt, G. (1988) *Oper. Res.* **36**, 493–513.

[3] Motwani, R. & Raghavan, P. (1995) *Randomized Algorithms* (Cambridge University Press, Cambridge).

[4] Alon, N. & Spencer, J. H. (1992) *The Probabilistic Method* (John Wiley & Sons, New York).

[5] Goemans, M. X. & Williamson, D. P. (1995) *J. ACM* **42**, 1115–1145.

[6] Goemans, M. X. (1997) *Math. Prog.* **79**, 143–161.

[7] Raghavan, P. & Thompson, C. D. (1987) *Combinatorica* **7**, 365–374.

[8] Arora, S. (1996) *Proc. 37th IEEE Symp. on Foundations of Computer Science*, 2–13.

[9] Mitchell, J. S. B. *SIAM J. Comput.*, to appear.

[10] Shmoys, D. B. (1995) in *Combinatorial Optimization*, eds. Cook, W., Lovász, L., and Seymour, P. D. (AMS, Providence), pp. 355–397.

[11] Hochbaum, D. S., ed. (1997) *Approximation Algorithms for NP-hard Problems* (PWS, Boston).