DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume 00, 0000

Computing Near-Optimal Solutions to Combinatorial Optimization Problems

DAVID B. SHMOYS

May 18, 1995

ABSTRACT. In the past few years, there has been significant progress in our understanding of the extent to which near-optimal solutions can be efficiently computed for *NP*-hard combinatorial optimization problems. This paper surveys these recent developments, while concentrating on the advances made in the design and analysis of approximation algorithms, and in particular, on those results that rely on linear programming and its generalizations.

In the past few years, there have been major advances in our understanding of performance guarantees for approximation algorithms for *NP*-hard combinatorial optimization problems. Most notably, after twenty-five years of essentially no progress, a new technique has been developed for proving that certain approximation algorithms are unlikely to exist. Partially in response to this development, there have also been significant recent advances in the design and analysis of approximation algorithms. In this survey, we will outline a few of the areas in which progress has been made, and suggest directions in which there is still interesting work to be done.

The central definition of this survey is that of a *ρ-approximation algorithm* for an optimization problem: a polynomial-time algorithm that delivers a feasible solution of objective function value within a factor of ρ of optimal. The study of approximation algorithms predates the theory of *NP*-completeness. Some early results, such as the proof due to Vizing (1964) that a graph always has

1991 *Mathematics Subject Classification.* Primary 90C27, 68Q25, 05C85; Secondary 68Q05, 90C05.

Research partially supported by NSF grant CCR-9307391, NSF PYI grant CCR-8896272 with matching support from UPS, Sun, Proctor & Gamble, and DuPont, and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550

an edge coloring with at most one more color than the maximum degree, were not even algorithmically stated, but still contain all of the ideas needed to give an interesting approximation algorithm. Graham (1966), in studying a particular scheduling problem, more explicitly stated the goal of analyzing an efficient heuristic from the perspective of its worst-case error bound, or more positively stated, its performance guarantee. The seminal work of Johnson (1974a, 1974b), reacting to the recent discovery of the theory of *NP*-completeness, crystallized the perspective that one approach to coping with the intractability of a problem is to design and analyze approximation algorithms.

Johnson (1974a,1974b) considered a variety of problems that have become, over the intervening years, some of the central questions of the area, including the bin-packing problem, maximum clique problem, the minimum vertex coloring problem, the maximum satisfiability problem, and the set covering problem. As we now understand much more clearly, these problems are all equivalent, in that they are *NP*-complete, and yet are very different in the extent to which good approximation algorithms exist for them. It is illuminating to consider the concluding questions posed by Johnson (1974b):

> Are there $O(\log n)$ coloring algorithms? Are there any clique finding algorithms better than $O(n^\epsilon)$ for all $\epsilon > 0$? ... What is it that makes algorithms for different problems behave in the same way? Is there some stronger kind of reducibility than the simple polynomial reducibility that will explain these results...?

While it must have been natural to suspect that some of these questions would be hard to answer, it is unlikely that the true difficulty of even the easiest of these questions was suspected at the time. For example, more than a decade elapsed before the 2-approximation algorithm given for the general maximum satisfiability problem was improved upon at all.

This survey is not intended to be a comprehensive account of the progress of those twenty years. Instead, we will focus on a few of the central issues and problems, and try to outline some of the main ideas that have been recently introduced. In particular, in describing the algorithmic advances, we will try to emphasize the role of linear programming and its generalizations in the design and analysis of approximation algorithms.

1. Probabilistic Proofs and Approximation Algorithms

Over the past several years, one of the fundamental threads of research in computational complexity theory has been to give randomized characterizations of deterministic complexity classes. For example, we now know that *PSPACE* is exactly the class of languages accepted by interactive proof systems and *NEXP* is exactly the class accepted by 2-prover interactive proofs. The culmination of this line of research was a new and surprising characterization of *NP*, which was in part motivated by the recently discovered connection between hardness of approximation results and randomized characterizations. We shall not attempt to give a complete description of the development of this important line of research, since this has been excellently surveyed by Johnson (1992). However, in this section we will give a brief overview of this area and its connection to the complexity of computing near-optimal solutions.

Let $|x|$ denote the length of a string x. A language L is in *NP* if there is a polynomial $p(n)$ and a polynomial-time verifying algorithm $V(x, y)$ that satisfies two conditions: (1) if $x \in L$, then there is a (proof) y for which $|y| \leq p(|x|)$ such that V outputs "yes"; (2) if $x \notin L$, then for every (proof) y for which $|y| \leq p(|x|)$ V outputs "no". More informally, the verifier can be convinced (by the correct proof) that $x \in L$ in such a way that he is never fooled. Extending work of Arora & Safra (1992), Arora, Lund, Motwani, Sudan, & Szegedy (1992) proved that the following is an equivalent definition: the verifier tosses $r = O(\log |x|)$ coins to determine $k = O(1)$ bit positions $i_l \in \{1, 2, \ldots, p(|x|)\}$; then, by examining x and only those positions of the proof the verifier answers "yes" or "no" in such a way that: (1) if $x \in L$, then there is a (proof) y for which $|y| \leq p(|x|)$ such that V outputs "yes" for all outcomes of the coin tosses; (2) if $x \notin L$, then for every (proof) y for which $|y| \leq p(|x|)$, V outputs "no" for more than half of the outcomes of the coin tosses. Since this verifier is allowed to use $O(\log n)$ random bits, and to access $O(1)$ positions of the proof, the class of languages recognized by such probabilistic verifiers is denoted *PCP*$(\log n, 1)$. Thus, the theorem of Arora, Lund, Motwani, Sudan, & Szegedy can be stated: *NP*=*PCP*$(\log n, 1)$. Due to its special properties, such a proof that $x \in L$ has sometimes been called a *holographic proof*.

Feige, Goldwasser, Lovász, Safra, & Szegedy (1991) observed an interesting connection between complexity class characterizations of this sort and approximation algorithms. To understand the essence of this connection consider the following *optimal proof problem*, defined by a probabilistic verifier V for an *NP*-complete language L: for any input x, find a "proof" so as to maximize the number of the 2^r coin tosses that lead the verifier to output "yes". If $x \in L$, then for the correct proof, the verifier accepts for all of its 2^r coin tosses. However, if $x \notin L$, then for any proof, fewer than 2^{r-1} coin tosses lead to acceptance. Hence, any 2-approximation algorithm for the optimal proof problem can be used to decide if $x \in L$ by merely checking if the proof found for input x has at least 2^{r-1} coin tosses that lead to acceptance. Since this checking can be done in polynomial time, we have a polynomial-time algorithm for L, and hence *P*=*NP*. In other words, no 2-approximation algorithm exists unless *P*=*NP*.

Of course, the optimization problem defined above is not a particularly natural one. It would be nice if we could use the same philosophy for specific combinatorial optimization problems. Feige et al. first applied this method to the *maximum clique problem*: given a graph $G = (V, E)$, find a subset S of pairwise adjacent vertices so as to maximize $|S|$. In fact, some of the strongest hardness-of-approximation results have been subsequently obtained for this problem: there exists an $\epsilon > 0$ such that no $|V|^{\epsilon}$-approximation algorithm exists unless P=NP. We shall defer a more detailed discussion of results for this problem until Section 9.

For some combinatorial optimization problems, it is possible to efficiently find solutions that are arbitrarily close to optimal. A *polynomial approximation scheme* for an optimization problem is a family of algorithms A_ρ, such that for each $\rho > 1$, A_ρ is a ρ-approximation algorithm. However, for many problems, there is no polynomial approximation scheme known, and until recently, no evidence to suggest that none exists.

Consider the *maximum 3-satisfiability problem* (Max 3-Sat): given a collection of clauses, each of which is the or of at most 3 literals, that is, Boolean variables and their negations, assign the values true or false to each of the n variables so as to maximize the number of clauses that are made true by this assignment. As a corollary of the fact that NP=$PCP(\log n, 1)$, Arora et al. obtained a striking result for this problem: they showed that no polynomial approximation scheme for the Max 3-Sat problem exists unless P=NP.

We next outline this result, which can be viewed as an encoding of the unnatural optimization problem discussed above as a Max 3-Sat problem. Consider the probabilistic verifier for some NP-complete language L; we will show that there is a constant $\delta > 0$ such that for any x, we can construct in polynomial-time a Max 3-Sat instance ϕ satisfying: (1) if $x \in L$, then ϕ has an assignment satisfying all of its clauses; and (2) if $x \notin L$, then any assignment does not satisfy a δ fraction of ϕ's clauses.

If we can compute such a ϕ, then we can prove that for any $\rho < 1/(1-\delta)$ no ρ-approximation algorithm exists, unless P=NP. Let m denote the number of clauses in ϕ. If $x \in L$, then the optimum value is m, and the approximation algorithm must find an assignment that satisfies more than $m(1-\delta)$ clauses. However, if $x \notin L$, then no such assignment exists. This yields a polynomial-time algorithm to decide if $x \in L$, and hence P=NP.

We next explain how to compute ϕ. The expression ϕ has a variable z_i for each bit i of a supposed holographic proof that $x \in L$. Each assignment to all of the Boolean variables z_i can be interpreted as a potential holographic proof: z_i is true if and only if the ith bit of the proof is 1. Let σ denote a possible outcome for the r coin tosses. We first construct an expression ϕ_σ that is satisfiable exactly when the verifier accepts, given that σ is the outcome of the coin tosses. Given σ, we can compute (in polynomial time) the k bits of the holographic proof that the verifier will examine. There are 2^k possible values for these bits, and each either causes the verifier to accept or reject. Since 2^k is a constant, we can compute, in polynomial time, which of these assignments causes the verifier to accept. Each of accepting assignments can be encoded by a Boolean expression that is the and of the appropriate k literals: for each bit i of the holographic proof examined, include z_i or $\neg z_i$ depending on whether the bit is 1 or 0, respectively. We can then encode all of the accepting values as a Boolean expression which is the or of terms, one term for each setting of the k bits that leads the verifier to accept. This expression can be converted to a 3-SAT expression ϕ_σ, albeit one with $K = 2^{2^k}$ clauses. However, since k is a constant, so is K. The MAX 3-SAT instance ϕ is the and of the expressions ϕ_σ for all possible σ. Since $r = O(\log n)$, there a polynomial number of possible coin tosses σ, and so ϕ has been computed in polynomial time.

We show that ϕ has the claimed properties. Clearly, if $x \in L$, then each expression ϕ_σ is made true by setting the variables according to the holographic proof for x; property (1) holds. If $x \notin L$ then for any potential holographic proof, more than half of the possible values for σ lead the verifier to reject. In other words, for any assignment of all Boolean variables z_i, more than half of the Boolean expressions ϕ_σ are not made true by this assignment. If ϕ_σ is false, at least 1 clause is not satisfied, which is a $1/K$ fraction of its clauses. Thus, for any assignment for Boolean variables z_i, at least a $1/(2K)$ fraction of ϕ's clauses are false. If we set $\delta = 1/(2K)$, we see that property (2) is also satisfied.

This construction is quite inefficient: the value of ρ for which one can prove that no ρ-approximation algorithm for MAX 3-SAT exists is extremely small. However, in subsequent work there has been substantial progress on improving this constant. Bellare, Goldwasser, Lund, & Russell (1993) showed that for any $\rho < 113/112$, if there exists a ρ-approximation algorithm, then *P*=*NP*. More recently, Bellare & Sudan (1994) further improved this value to 74/73. We shall discuss the complementary algorithmic results for this problem in Section 3.

Although the result of Arora et al. for the MAX 3-SAT problem is of great importance in and of itself, the consequences of this work are greatly magnified in conjunction with earlier work of Papadimitriou & Yannakakis (1991). Papadimitriou & Yannakakis observed that for many simple optimization problems, the state of the art circa 1991 was roughly the following: virtually any naive algorithm delivers a solution of value within a constant, most often 2, of optimal; no algorithm with better performance guarantee is known; there is no known complexity-theoretic evidence to support the supposition that no polynomial approximation scheme exists. Among the problems exhibiting this behavior were the maximum 3-satisfiability problem, the minimum vertex cover problem, and the maximum cut problem. In view of the success of the theory of *NP*-completeness, they sought an analogous notion of completeness, where proving that no polynomial approximation scheme exists for one complete problem implies the same result for each complete problem. Not only did they derive such a theory, which they called *MAXSNP*-completeness, but the result of Arora et al. completed the agenda, since it proved that no polynomial approximation scheme exists (assuming *P*≠*NP*) for a *MAXSNP*-complete problem, the maximum 3-satisfiability problem.

In the remainder of this section, we briefly sketch the main components of the framework proposed by Papadimitriou & Yannakakis (1991). The first element is the notion of reducibility from problem Π_1 to Π_2; here we wish to define a notion that allows us to conclude that if Π_1 does not have a polynomial approximation scheme, then neither does Π_2. An *L-reduction* consists of two polynomial-time computable functions: f maps an instance x of Π_1 into an instance of $f(x)$ of Π_2, whereas g maps a feasible solution s for the instance $f(x)$ of Π_2 into a feasible solution $g(s)$ for x; let $OPT_{\Pi_i}(I)$ denote the optimal value of Π_i for a given input I; we require that there be a constant α such that $OPT_{\Pi_2}(f(x)) \leq \alpha OPT_{\Pi_1}(x)$; finally, we require that there be a constant β such that the absolute value of the deviation from optimality of $g(s)$ is at most β times the absolute deviation from optimality of s.

We show that if there exists a polynomial approximation scheme for Π_2, then there must be one for Π_1 as well. Equivalently, we wish to show that, using a polynomial approximation scheme for Π_2, we can derive, for any $\delta > 0$, a polynomial-time algorithm for Π_1 where the absolute deviation from optimality is at most δOPT_{Π_1}. Given an input x, we compute $f(x)$, apply a $(1+\epsilon)$-approximation algorithm to this instance of Π_2 to get a feasible solution s, and finally compute $g(s)$, which is the output. The absolute deviation from optimality of s is at most $\epsilon OPT_{\Pi_2}(f(x))$, which is at most $\alpha\epsilon OPT_{\Pi_1}(x)$. Hence, the absolute deviation from optimality of $g(s)$ is at most $\beta\alpha\epsilon OPT_{\Pi_1}(x)$. If we set $\epsilon = \delta/(\alpha\beta)$, we have attained the desired goal.

Following the exposition in Papadimitriou (1994), we first define a more restrictive complexity class, *MAXSNP*$_0$. A maximization problem Π is in the class *MAXSNP*$_0$ if it can be formulated as

$$\max_S |\{(z_1, \ldots, z_k) \in V^k : \phi(G_1, G_2, \ldots, G_m, S, z_1, \ldots, z_k)\}|,$$

where the input to Π is a set of relations $G_1, \ldots, G_m$ over V, a finite universe, and ϕ is a quantifier-free first-order expression involving the variables z_i, the input, and an r-ary relation S; in words, the optimization problem is to find the relation that maximizes the number of assignments for z that cause ϕ to hold. This definition was motivated by the characterization of *NP* by Fagin (1974) as all languages that can be formulated in existential second-order logic.

To illustrate this definition, consider the *maximum directed cut problem*: given a directed graph $G = (V, A)$, find a set $S \subseteq V$, so as to maximize the number of edges (u, v) with $u \in S$ but $v \notin S$. To cast this problem in the form given above, we let the vertex set V be the universe, let $G(u, v)$ be the binary relation over V that holds exactly when $(u, v) \in A$, and let S be a unary relation over V; we wish to

$$\max_S |\{(z_1, z_2) \in V^2 : G(z_1, z_2) \wedge S(z_1) \wedge \neg S(z_2)\}|.$$

To complete the definition of *MAXSNP*, we say that any optimization problem Π_1 is in *MAXSNP* if there exists a problem $\Pi_2 \in$ *MAXSNP*$_0$ such that Π_1 L-reduces to Π_2. A problem is *MAXSNP*-hard if every problem in *MAXSNP* is L-reducible to it. Finally, a problem is *MAXSNP*-complete if it is in *MAXSNP* and it is *MAXSNP*-hard.

It is not hard to show that the MAX 3-SAT problem is in $MAXSNP_0$, and hence in *MAXSNP*. Papadimitriou & Yannakakis (1991) showed that it is also complete. There is now a respectably long list of problems known to be *MAXSNP*-hard. In addition to the problems mentioned above, the maximum acyclic subgraph problem, the traveling salesman problem, the maximum clique problem, the minimum Steiner tree problem, and the maximum superstring problem (and in many cases, very restrictive cases of each) are known to be *MAXSNP*-hard. Many of these problems will be addressed in detail later in this survey. Recently, Khanna, Motwani, Sudan, & Vazirani (unpublished manuscript) have argued that this definition for *MAXSNP* is the most natural one, since it has as a consequence that *MAXSNP* is exactly the class of optimization problems that are approximable within some constant factor.

2. A prototypical example: the vertex cover problem

In this survey, we will emphasize the role of linear programming in the design and analysis of approximation algorithms. In this section, we will study one problem, the weighted vertex cover problem, since it serves as a good example for many of the ways in which linear programming arises in this area. The *minimum-weight vertex cover problem* is as follows: we are given a graph $G = (V, E)$ and a weight $w(v) \geq 0$ for each vertex $v \in V$, and we wish to select a subset of vertices $S \subseteq V$ such that $u \in S$ or $v \in S$ for each edge $(u, v) \in E$, so as to minimize the total weight of S, $w(S) := \sum_{v \in S} w(v)$. Papadimitriou & Yannakakis (1991) showed that the vertex cover problem is *MAXSNP*-hard, even if each weight is 1 and the degree of each vertex is bounded by a constant k.

The following is a simple integer programming formulation of the minimum-weight vertex cover problem:

$$\text{(1)} \qquad \text{minimize} \sum_{v \in V} w(v)x(v)$$

subject to

$$\begin{aligned} &\text{(2)} \qquad & x(u) + x(v) &\geq 1, \qquad && \text{for each } (u, v) \in E, \\ &\text{(3)} \qquad & x(v) &\in \{0, 1\}, \qquad && \text{for each } v \in V. \end{aligned}$$

Let $OPT_{VC}(I)$ denote the optimal value for a given input I. There is a very simple (to explain) algorithm that always delivers a solution of weight at most $2OPT_{VC}(I)$. First, solve the linear relaxation of this integer program; i.e., replace equation (3) by

$$\text{(4)} \qquad x(v) \geq 0, \qquad \text{for each } v \in V.$$

Let x^* denote the optimal solution to this linear program. Next, construct an integer solution $\bar{x}$ by rounding x^* in the most natural way: for each $v \in V$, set

$$\bar{x}(v) = \begin{cases} 1 & \text{if } x^*(v) \geq .5, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

We claim that $\bar{x}$ is a feasible solution to the integer program (1)-(3) of weight at most

$$2 \sum_{v \in V} w(v) x^*(v) \leq 2OPT_{VC}(I).$$

To see that this solution is feasible, observe that the constraint (2) for edge (u, v) ensures that at least one of $x^*(u)$ and $x^*(v)$ is at least .5; hence, at least one of $\bar{x}(u)$ and $\bar{x}(v)$ will be set to 1. Furthermore, the rounding has the property that $\bar{x}(v) \leq 2x^*(v)$ for each $v \in V$, and so we obtain the fact that the weight of the cover corresponding to $\bar{x}$ is at most $2OPT_{VC}(I)$.

This algorithm is the simplest example of a *rounding procedure*: we find near-optimal solutions for a particular problem by giving an integer programming formulation, solving its linear relaxation, and then applying an efficiently computable rounding procedure to produce the desired integer solution.

This is a very simple result, and in its simplicity this algorithm ignores some of the interesting underlying structure of the problem. When we solve the linear relaxation we would typically compute an extreme point of the feasible region, and such extreme points sometimes have a much simpler structure that might help in designing the rounding procedure. For this problem, it is well known that any extreme point x^* has the property that for each $v \in V$, $x^*(v) \in \{0, .5, 1\}$ (see, for example, Nemhauser & Trotter (1975), Balinski & Spielberg (1969), and Lorentzen (1966)).

Unfortunately, this fact does not improve the performance guarantee of our rounding procedure. The additional structure does implicitly help improve the efficiency of our algorithm, since it is not hard to show that the problem of finding the minimum $\{0, .5, 1\}$-solution can be reduced to the maximum flow problem; consequently, the linear program can be solved much more efficiently than an arbitrary linear program. The connection between these relaxations and approximation algorithms for the minimum-weight vertex cover problem was first observed by Hochbaum (1982), who gave the first 2-approximation algorithm for the problem.

The linear relaxation can also be exploited in a less explicit way to design even more efficient algorithms. Suppose that we design a polynomial-time algorithm that simultaneously constructs an integer primal solution and a feasible solution to the dual linear program. Since the value of a feasible dual solution gives a lower bound on both the optimal linear and integer primal solution values, if we also know that the value of the primal integer solution is no more than ρ times the value of the dual solution, then the algorithm is a ρ-approximation algorithm. We will call such an algorithm a *primal-dual algorithm*.

Completing our example of the vertex cover problem, we shall give a simple primal-dual algorithm, which is due to Bar-Yehuda & Even (1981). To begin, we formulate the dual linear program:

$$\text{maximize} \sum_{e \in E} y(e) \tag{6}$$

subject to

$$\sum_{e \in \delta(v)} y(e) \quad \leq \quad w(v), \qquad \text{for each } v \in V, \tag{7}$$

$$y(e) \quad \geq \quad 0, \qquad \text{for each } e \in E, \tag{8}$$

where $\delta(v)$ denotes the set of edges $\{e \in E : e = (u, v)\}$. For ease of discussion, let a *w-packing* be an assignment of values to edges so that the total value of the edges incident to v is at most $w(v)$. The dual is then to find a w-packing of maximum total value.

It is instructive to consider the specialization of this problem when $w(v) = 1$, for each $v \in V$; that is, the unweighted case. If we just consider integer dual solutions, then we see that the constraints restrict us to $\{0,1\}$-solutions, which correspond to matchings (i.e., subsets of edges such that no two edges have a common endpoint). An easy 2-approximation algorithm for the unweighted vertex cover problem is as follows: find a maximal matching M and output the vertex cover S consisting of all endpoints of edges in M. (Note that M is only a *maximal* matching, i.e., for any $e \in E - M$, $M \cup e$ is not a matching.) To see that S is a vertex cover, observe that if there is an edge e which is not covered by one of its endpoints, then e can be added to M to form a matching, which is a contradiction. The matching is a feasible dual solution, and the solution S output is exactly a factor of 2 bigger than it. Hence, this is a 2-approximation algorithm. One does not need linear programming duality to deduce that the size of the matching is a lower bound on the optimum size of a vertex cover: each edge in the matching must be covered by a distinct vertex. Observe that we did not even construct the optimal dual solution, even though that is possible in polynomial time; the analysis did not require that. In fact, we really wish to construct as small a maximal matching as possible.

Consider again the weighted version. We wish to design an algorithm analogous to the one discussed above in the unweighted case. Call a node v *tight* for a given feasible dual solution y if its corresponding constraint (7) is satisfied with equality. Initialize the cover S being constructed to the empty set, the dual variable $y(e)$ to 0, for each edge $e \in E$, and repeat the following step until all edges of the graph have been deleted: choose any remaining edge $e = (u, v)$ and increase its value $y(e)$ as much as possible while maintaining the feasibility of the current dual solution (i.e., until the first of its endpoints becomes tight); add the tight node(s) to the cover S; delete the tight node(s) and their incident edges.

There are two main steps to analyze this algorithm: proving that the set constructed is indeed a vertex cover, and proving that its weight is at most twice the value of the dual solution constructed. To show that S is a vertex cover at the end of the algorithm, one need only note that an edge is only deleted when one of its endpoints is tight, and hence in the cover. Suppose that whenever a variable $y(e)$ is increased by k, one pays k dollars to each of the endpoints of e. Thus, the total amount paid to the nodes of G is exactly twice the value of the dual solution. Observe that whenever a node v is put in the cover, there must be exactly $w(v)$ dollars paid to it, since the constraint (7) is tight. We can think of the algorithm as putting a node v in the cover only when its price $w(v)$ is totally paid for. Hence, the weight of the nodes placed in the cover is at most the total amount paid to all nodes, which is twice the value of the dual solution. We have shown that the primal-dual algorithm is a 2-approximation algorithm. Quite surprisingly, for any constant $\rho < 2$, no ρ-approximation algorithm is known, even if all of the weights are 1. It remains a challenging open problem to find such an algorithm.

3. Randomized and Derandomized Rounding

The definition of the complexity class *MAXSNP* was motivated by the overwhelming lack of progress on designing approximation algorithms for a handful of simple combinatorial optimization problems, such as the vertex cover problem. For several of these problems, the most naive algorithms remained the ones with best known performance guarantee. Fortunately, for a few of these problems there has been some progress, even substantial progress, made in the past few years.

One paradigm for designing algorithms that has received a great deal of attention recently is to design a randomized algorithm first, and then "derandomize it" by, in some sense, simulating the role of the randomization in critical places in the algorithm. An early example of this approach is implicit in the work of Johnson (1974b) on the *maximum-weight satisfiability problem*, which is a weighted generalization of the problem considered in Section 1: given a collection of clauses, each of which is the or of (any number of) literals, as well as a weight w_i for each clause C_i, assign the values true or false to each of the n variables so as to maximize the total weight of the clauses that are made true by this assignment.

First consider the following trivial randomized algorithm for this problem: for each Boolean variable v_j, independently set v_j to be true with probability 1/2, and set it to be false with probability 1/2. By the linearity of expectation, the expected total weight of the resulting assignment is just the sum over all clauses of the expected weight contributed by each clause. Consider a clause C_i, and suppose that it has k distinct literals. The random assignment makes this clause true with probability at least $1 - 2^{-k}$. Thus, the expected contribution of this clause to the weight of the assignment is at least $(1-2^{-k})w_i$. Consequently, the expected weight of the random assignment for an instance in which each clause has at least k literals is at least $(1-2^{-k})\sum_i w_i$. This is clearly within a factor $1/(1-2^{-k})$ of optimal, since the optimum is at most $\sum_i w_i$. Since k is at least 1, this factor is at most 2.

We will outline a simple approach to obtain the same performance guarantee deterministically. What we wish to compute is an assignment of total weight that is at least the expectation. This can be done by the *method of conditional probabilities*. Observe that for any partial assignment of the variables, one can easily compute the conditional expectation of the weight of the assignment formed by completing the partial one by a random assignment chosen independently and uniformly. The deterministic algorithm works by assigning an additional variable in each iteration. Suppose that $v_1, \ldots, v_j$ are already assigned. We compute the conditional expectation of this partial assignment first with v_{j+1} also set to false, and then again with it set to true. If we set v_{j+1} according to which of these values is greater, then the conditional expectation at the end of iteration $j+1$ is at least the conditional expectation at the end of iteration j. This implies that the complete assignment derived has weight at least the original (unconditional) expectation. Thus, this is a 2-approximation algorithm.

The analysis described above also implies that if the input does not contain any clause of length 1, then the algorithm finds an assignment of weight within a factor of 4/3 of the optimum. Yannakakis (1992) gave an algorithm that essentially eliminated all clauses of length 1 in a way that preserves the performance guarantee, and thereby gave a 4/3-approximation algorithm. Goemans & Williamson (1993) later gave a much simpler algorithm with the same performance guarantee. Their algorithm is based on the idea that the randomized algorithm might perform better if, with a little work, one computes for each variable v_j, a more refined probability choice p_j with which to set that variable true.

The more refined probabilities are computed by solving the linear relaxation of a natural $\{0,1\}$-integer programming formulation of the maximum-weight satisfiability problem. We introduce a $\{0,1\}$-variable x_j for each Boolean variable v_j and a $\{0,1\}$-variable z_i for each clause C_i; a binary variable set to 1 corresponds to a Boolean variable set to true. For each clause C_i, let T_i index the set of Boolean variables that occur unnegated in C_i, and let F_i index the set of Boolean variables that occur negated in C_i. Then, we wish to ensure that a clause variable z_i is set to 1 only if one of the literals in that clause is true:

$$z_i \leq \sum_{j \in T_i} x_j + \sum_{j \in F_i} (1 - x_j).$$

By letting the objective function be to maximize $\sum_i w_i z_i$, we obtain an integer programming formulation of this problem. Let (x^*, z^*) denote the optimal solution for the linear relaxation of this integer program in which all variables are instead bounded between 0 and 1. Independently set each variable v_j true with probability $p_j = x_j^*$ and false with probability $1 - p_j$. Raghavan & Thompson (1987) introduced and analyzed this approach of rounding a fractional solution to a $\{0,1\}$-integer solution by interpreting the fractions as probabilities and performing such a *randomized rounding*, albeit in the context of a VLSI routing problem.

We can view $w_i z_i^*$ as the contribution of clause C_i to the weight of the fractional assignment. A straightforward computation shows that if C_i has k literals, then the expected contribution of clause C_i in the assignment found by randomized rounding is at least $(1-(1-\frac{1}{k})^k)w_i z_i^*$. This quantity is at least $(1-1/e)w_i z_i^*$, and hence we expect to obtain a solution of weight at least $(1-1/e)\sum_i w_i z_i^*$, where $\sum_i w_i z_i^*$ is an upper bound on the optimal weight. Applying the method of conditional probabilities, we obtain an $e/(e-1)$-approximation algorithm.

To match the performance guarantee of the result of Yannakakis (1992), Goemans & Williamson (1993) employ a frequently used technique in the design and analysis of approximation algorithms: combine two complementary algorithms which have disjoint sets of bad cases, by running both and choosing the better solution. Observe that the two randomized algorithms already described appear to have complementary properties: the linear programming approach works best for short clauses, whereas the uniform approach works best for long ones. More precisely, if we first toss a fair coin to select which of the two algorithms to run, then the expected contribution of a clause C_i with k literals is at least

$$[(1-2^{-k}) + (1-(1-(1/k))^k)]w_i z_i^*/2 \geq 3w_i z_i^*/4.$$

Thus, the expected value of at least one of the two random assignments is at least 3/4 of the optimal value of linear programming relaxation. This implies that if we run both algorithms and select the better solution, then we obtain a 4/3-approximation algorithm.

Goemans & Williamson (1993) also show that a 4/3-approximation algorithm can be obtained by a single rounding of the linear programming optimal solution; rather than interpreting the fractional solution as probabilities directly, they instead show how to map fractional solutions to probabilities that, when used for randomized rounding, are expected to yield good solutions. In contrast, the strongest currently known hardness result, due to Bellare & Sudan (1994), is that approximating this problem within 74/73 is *NP*-hard, even if all weights are 1.

Linear programming relaxations have played a central role in the design and analysis of approximation algorithms. However, recently there has been striking success in basing approximation algorithms on more general mathematical programming tools, such as convex programming. In the *maximum-weight cut problem*, the input consists of an n-vertex graph $G = (V, E)$, where each edge e has an associated nonnegative weight $w(e)$, and the aim is to find a set $S \subseteq V$, so as to maximize the value of $\sum_{e=(u,v):u\in S, v\notin S} w(e)$. Throughout the remainder of this survey, we shall let $\delta(S)$ denote the set of edges in the *cut* defined by S, $\{e = (u, v) : u \in S, v \notin S\}$.

Until recently, the best constant performance guarantee known for this problem was 2, due to Sahni & Gonzalez (1976). Their algorithm can also be viewed as a derandomized randomized algorithm. In this case, the randomized algorithm is, for each vertex v, to include $v \in S$ independently with probability 1/2. The expected value of the cut is the sum over all edges $e = (u, v)$ of the product of its weight $w(e)$ and the probability that exactly one of u and v is in S. Since this probability is exactly 1/2, the expected weight of the cut found is exactly $\sum_{e \in E} w(e)/2$. Since the total weight of the edges in the graph is clearly an upper bound on the optimal value, the expectation is within a factor of 2 of the weight of the maximum cut. By applying the method of conditional probabilities, we obtain the following algorithm: in iteration $j = 1, \ldots, n$, we decide whether or not to put $v_j \in S$: compute the additional weight added to the current cut if v_j is added to S or not added to S, and make the decision for v_j depending on which is greater.

One reformulation of the maximum-weight cut problem can be stated as follows: for each vertex $v \in V$, assign it a value $x(v) \in \{+1, -1\}$, to indicate whether v is in S or not; given this assignment, the contribution of edge $e = (u, v)$ to the objective function is $w(e)(1 - x(u)x(v))/2$.

Goemans & Williamson (1994) gave a randomized approximation algorithm for the maximum-weight cut problem based on the following relaxation: for each vertex v, assign it an n-dimensional unit-length vector $\vec{x}(v)$ so as to maximize $\sum_{e=(u,v) \in E} w(e)(1 - \vec{x}(u) \cdot \vec{x}(v))/2$, where $\vec{x} \cdot \vec{y}$ denotes the inner product of $\vec{x}$ and $\vec{y}$. The algorithm is quite simple: find an optimal solution $\vec{x}^*$ to this relaxation, and select an n-dimensional unit-length vector $\vec{r}$ uniformly at random; put those vertices v in S for which $\vec{x}^*(v) \cdot \vec{r} \geq 0$. More geometrically, we are bisecting the n-dimensional unit sphere by a hyperplane through the origin, and thereby separating the unit-length vectors computed by the relaxation into two sets that determine a cut.

The analysis of this algorithm is also remarkably simple. Let the weight of the cut found by the algorithm be the random variable W. We wish to estimate the value of the expectation of W. By linearity of expectation, this is $\sum_{e=(u,v) \in E} w(e) Pr[e \in \delta(S)]$. For each $e = (u, v)$, we can compute $Pr[e \in \delta(S)]$ by observing that u and v end up on opposite sides of the cut exactly when the random hyperplane defined by $\vec{r}$ lies "in between" the vectors $\vec{x}^*(u)$ and $\vec{x}^*(v)$; hence this probability is proportional to the angle defined by these vectors, and is $\arccos(\vec{x}^*(u) \cdot \vec{x}^*(v))/\pi$. A straightforward calculation shows that for any y such that $-1 \leq y \leq 1$, $\arccos(y)/\pi \geq \alpha(1 - y)/2$, where $\alpha = \min_{0 < \theta \leq \pi} (2/\pi)(\theta/(1 - \cos\theta))$. Hence, the expected weight of the cut found is at least

$$\sum_{e=(u,v) \in E} \alpha w(e)(1 - \vec{x}^*(u) \cdot \vec{x}^*(v))/2.$$

In other words, the expected weight is at least an α fraction of the optimal value of this non-linear relaxation, and hence at least an α fraction of the optimal cut value. It turns out that α is roughly .87856, and so this is within a factor of 1.1393 of the optimal.

To compute an optimal solution to the relaxation, we can introduce one variable $y(u,v)$ for each inner product of unit-length vectors $x(u) \cdot x(v)$. It is not hard to see that y is of this form exactly when the matrix $Y = (y(u,v))$ is symmetric positive semidefinite and has $y(v,v) = 1$ for each $v \in V$. We wish to maximize a linear objective function in Y subject to these constraints. This type of mathematical programming problem is usually referred to as a positive semidefinite program, and can be solved within an additive error of ϵ in time polynomial in the size of the input and $\log(1/\epsilon)$ by, for example, the ellipsoid algorithm. Although the details are more complicated, this algorithm can also be derandomized by the method of conditional expectations, and so this yields a 1.1393-approximation algorithm for the weighted maximum cut problem.

Similar relaxations can also be formulated for the special case of the maximum-weight satisfiability problem in which each clause has length at most 2. Prior to the work of Goemans & Williamson (1994), no approximation algorithm with a performance guarantee superior to that for the general case was known. They showed that there is a 1.1393-approximation algorithm based on this approach. Feige & Goemans (private communication) have further strengthened the relaxation to give a 1.075-approximation algorithm. Both of these results can also be applied to yield an incremental improvement on the 4/3 performance guarantee known for the general case of this problem.

One curious phenomenon should be noted for each of the problems (and their special cases) that we have studied thus far. The best known performance guarantee for the weighted case exactly matches that for the unweighted one. It would be interesting to know if there is some underlying explanation for this.

4. Centers, Tours, and Steiner Trees

For some problems, a simple ρ-approximation algorithm can be seen to be best possible, in the sense that, for any $\rho' < \rho$, no ρ'-approximation algorithm exists unless *P=NP*. One such problem is the *minimum p-center* problem. The input consists of an integer p, and an $n \times n$ symmetric distance matrix $C = (c_{ij})$ that satisfies the triangle inequality (i.e., $c_{ij} + c_{jk} \geq c_{ik}$, for each i, j, k) where c_{ij} specifies the distance between each pair of points i, j in $\{1, 2, \ldots, n\}$: for any selection $S \subseteq V$ of designated centers, the *radius* of S is the minimum distance r such that each point is within r of some point in S; the aim of the p-center problem is to select a set S of size p so as to minimize its radius.

Hochbaum & Shmoys (1985) gave a simple 2-approximation algorithm for the p-center problem. First consider a decision version of the problem, where a radius r is also given, and you wish to decide if there exist p points to select of radius r. Clearly, by performing a bisection search over the $O(n^2)$ possible values of the optimal value, one can use an algorithm that solves the decision problem to find the optimal radius. We give a relaxed version of such a decision algorithm, that either outputs correctly that no solution of radius r exists, or else finds a set of p points of radius $2r$. By performing a bisection search for an appropriate value of r, such a relaxed decision procedure immediately yields a 2-approximation algorithm. A different 2-approximation algorithm for this problem was also given by Dyer & Frieze (1985).

Consider the following algorithm: repeatedly choose one of the remaining points i, add i to S, and delete all points whose distance from i is at most $2r$. At the end of this process, if the size of the set S exceeds p, output that no set of size p and radius r exists, and otherwise output S. It is clear that if S is output, then it has radius at most $2r$, since the algorithm was designed so that this must happen. We must only show that if there is a set S^* of size at most p and radius at most r, then the algorithm outputs a set S of size at most p. Suppose there is such a set S^*, and assign each point to its closest point in S^*. Thus, we have partitioned the points into at most p parts, $S_1^*, \ldots, S_p^*$. The algorithm can select at most one point i from each part S_j^*, since the selection of i causes all remaining points in S_j^* to be deleted. Hence, at most p points are selected by the algorithm.

Hsu & Nemhauser (1979) observed that if there exists a ρ-approximation algorithm with $\rho < 2$, then $P{=}NP$. To see this, consider the *dominating set problem*, which is NP-complete: given a graph $G = (V, E)$ and an integer p, decide if there exists a set $S \subseteq V$ of size p, such that each vertex is either in S, or adjacent to a vertex in S. Given an instance of the dominating set problem, we can define an instance of the p-center problem by setting the distance between adjacent vertices to 1, and non-adjacent vertices to 2: there is dominating set of size p if and only if the optimal radius for this p-center instance is 1. Furthermore, any ρ-approximation algorithm with $\rho < 2$ must always produce a solution of radius 1 if such a solution exists, since any solution of radius $\rho < 2$ must actually be of radius 1.

For combinatorial optimization problems defined on a distance metric, it is of interest to consider specific metrics, such as the L_1, L_2, or L_∞ norms, and ask whether superior performance in such a special case is possible. For each of these cases, it is known that no ρ-approximation algorithm for the p-center problem exists unless $P{=}NP$, for all $\rho < \rho_0$, where ρ_0 is a particular value less than 2 that depends on the metric; for example, for the Euclidean case, $\rho_0 = \sqrt{2+\sqrt{3}}$ (Mentzer (unpublished manuscript), Feder & Greene (1988)). However, for none of these cases is a ρ-approximation algorithm with $\rho < 2$ known, and we conjecture that such algorithms exist.

A natural generalization of the problem is to relax the restriction that the distance matrix be symmetric. This turns out to be a non-trivial generalization, and essentially nothing is known about performance guarantees for this extension.

In the remainder of the section, we will discuss two other optimization problems defined on points in a given distance metric: the traveling salesman problem and the minimum Steiner tree problem. For each, we will discuss the known results in the symmetric case with triangle inequality, discuss improvements for special metrics, and then consider the asymmetric case.

In the *traveling salesman problem* (TSP), the aim is to find a cyclic permutation of the points π, so as to minimize $\sum_{i=1}^{n} c_{i\pi(i)}$; let $OPT_{TSP}(C)$ denote the optimal tour length. Rosenkrantz, Stearns, & Lewis (1977) showed that the following is a 2-approximation algorithm: maintain a tour T on a subset of the points, initially a single point; iteratively find the point $i \notin T$ that is closest to some point $j \in T$, and insert i into T after j. The analysis is based on the observation that the algorithm uses a sequence of pairs (i, j) that exactly mimics Prim's minimum spanning tree algorithm, and it is not hard to show that the increase in tour cost in each iteration is at most $2c_{ij}$. Since the optimal value of the minimum spanning tree is at most $OPT_{TSP}(C)$, the algorithm is a 2-approximation algorithm.

Christofides (1976) gave the only known improvement to this algorithm. First consider the following alternative 2-approximation algorithm: find a minimum spanning tree, and take two copies of each edge to form an Eulerian graph; find an Eulerian tour, and order the vertices as they are first encountered on this tour. By the triangle inequality, the cost of the cyclic permutation found is no more than the total cost of the Eulerian graph, which is clearly at most $2OPT_{TSP}(C)$. This "double-tree" algorithm can be improved by augmenting a minimum spanning tree T to a less expensive Eulerian graph. Christofides observed that a minimum-cost perfect matching M on the vertices of odd degree in T has cost at most $OPT_{TSP}(C)/2$, since an optimal tour on those vertices can be partitioned into two perfect matchings. Hence, the cyclic permutation computed from an Eulerian tour of the union of T and M has cost at most $3/2(OPT_{TSP}(C))$.

Papadimitriou & Yannakakis (1993) gave the first evidence that obtaining near-optimal solutions for the TSP is hard, by showing that the TSP is *MAXSNP*-hard, even when restricted to symmetric instances with each $c_{ij} \in \{1, 2\}$. Therefore, a corollary of the result of Arora, Lund, Motwani, Sudan, and Szegedy (1992) is that no polynomial approximation scheme for the TSP with triangle inequality exists, unless *P=NP*. Of course, this does not imply that a similar result holds for any of the specialized metrics discussed above: L_1, L_2, or L_∞. Remarkably, no stronger performance guarantees are known for these special cases, nor is there evidence that no polynomial approximation scheme exists. It is possible that we simply do not have the technical dexterity in manipulating L-reductions that we have acquired for ordinary polynomial-time reductions. If this is true, one might expect that completeness results will be proven for these special metrics. However, it would be very nice if it were possible to take advantage of the additional geometric structure so as to design a polynomial approximation scheme for these cases.

The asymmetric TSP (with triangle inequality) appears to be substantially harder than the symmetric case. The best known algorithm, due to Frieze, Galbiati, & Maffioli (1982), works by finding an optimal assignment (i.e., find a permutation π that minimizes $\sum_{i=1}^{n} c_{i\pi(i)}$) with each $c_{ii} = \infty$: if π is cyclic, then this solution is output; otherwise, one node is selected from each cycle, and the algorithm is called recursively on this subset of nodes, which is of size at most $n/2$. Since the optimal assignment for any subset is no more than the complete optimal tour, and the depth of recursion is $\log_2 n$, this is a $\log_2 n$-approximation algorithm. It is tempting to conjecture that this is asymptotically optimal, but the optimists among us continue to hope (and to try to prove) otherwise.

In considering the symmetric TSP, while no approximation algorithm better than Christofides' algorithm is known, there is reason to believe that better algorithms do exist. If we consider the input to be the complete graph $K_n = (V, E)$ where each edge e has cost $c(e)$, then we can formulate the following linear relaxation of the problem:

$$\text{minimize} \sum_{e \in E} c(e)x(e) \tag{9}$$

subject to

$$\sum_{e \in \delta(v)} x(e) = 2, \quad \text{for each } v \in V, \tag{10}$$

$$\sum_{e \in \delta(S)} x(e) \geq 2, \quad \text{for each } S \subseteq V, S \neq \emptyset, \tag{11}$$

$$x(e) \geq 0, \quad \text{for each } e \in E. \tag{12}$$

Wolsey (1980) showed that the value of this linear program is always at least $2/3$ of the optimum TSP value. Alternate proofs of this fact are given by Goemans & Bertsimas (1993) and Shmoys & Williamson (1990). The latter is based on showing that Christofides' algorithm always produces a tour of length within a factor of $3/2$ of this relaxation. In contrast to Christofides' algorithm, the analysis of the ratio between this linear optimum and the integer optimum is not known to be tight. In fact, it is widely conjectured that the true worst-case ratio is $4/3$.

Motivated by this example, we define a *ρ-estimation algorithm* to be a polynomial-time algorithm that produces a value that is at most the optimal value, and is always within a factor of ρ of it. As opposed to algorithms that find optimal solutions or values, there is no known (polynomial-time) equivalence between finding near-optimal solutions and estimating the optimal value with the same performance guarantee. All known non-existence results about approximation algorithms are actually for the non-existence of an estimation algorithm. It would be interesting to show, for example for the TSP, that there is a ρ-estimation algorithm, and yet for some $\rho' > \rho$, no ρ'-approximation algorithm exists unless *P=NP*. Or alternatively, it would be remarkable to find a kind of self-reducibility that allows us to show that the two types of guarantees are equivalent.

In the minimum Steiner tree problem, we are given a subset of *terminals* $T \subseteq V$ along with a metric defining the distance between any two points in V. The objective is to choose a connected subgraph that spans the terminals of minimum total cost. The set V need not be finite, such as in the Euclidean case where V is the entire Euclidean plane. There is a simple 2-approximation algorithm known for an arbitrary metric. Compute a minimum spanning tree on the complete graph G_T defined by the terminal set T; that is, do not use the Steiner points $V - T$ at all. To see that this is a 2-approximation algorithm, we bound the length of the minimum spanning tree in G_T by showing that G_T has a connected spanning subgraph of length at most the twice the cost of the optimal Steiner tree. In fact, we construct a TSP-like tour of this cost: we convert the optimal Steiner tree into a tour of its terminals by using a "double-tree" traversal.

Zelikovsky (1993) provided the breakthrough that showed how to make non-trivial use of Steiner points in designing an approximation algorithm with a better performance guarantee. Take any three terminals $u, v, w \in T$, and compare the length of the minimum spanning tree on G_T to the sum of the length of the minimum Steiner tree connecting just u, v, and w, and the length of the minimum spanning tree once u, v, and w have been contracted to one point. If the former is larger, then clearly we can construct a better Steiner tree by combining the 3-terminal Steiner tree with the minimum spanning tree for the smaller problem. The difference between these two values is called the *gain* of $\{u, v, w\}$. Zelikovsky's algorithm repeatedly contracts triples of terminals, choosing the triple with the greatest gain, until no triple with positive gain exists. The resulting Steiner tree is constructed from the minimum spanning tree on the remaining terminals, combined with the Steiner trees associated with each contraction along the way. The analysis of the algorithm consists of two parts: that an optimal sequence of contractions yields a Steiner tree of value within a factor of $5/3$ of the optimal Steiner tree length; and the sequence of contractions found by this greedy approach has a total gain that is at least half of the optimal total gain. Consequently, this algorithm is an $11/6$-approximation algorithm.

For Zelikovsky's algorithm to run in polynomial time, it is necessary to compute a minimum Steiner tree on 3 points in polynomial time in the given metric. While there are metrics for which this is not possible, for each of the metrics given above, or any setting for which V is finite, this computation can be done in polynomial time. Berman & Ramaiyer (1992) extend Zelikovsky's scheme, incorporating optimal Steiner trees on more points to yield improved performance guarantees. For example, by using 4-tuples it is possible to derive a $16/9$-approximation algorithm.

In one of the earliest papers on computational aspects of the minimum Steiner tree problem, Gilbert & Pollak (1968) conjectured that the minimum spanning tree was always of length within a factor of $2/\sqrt{3}$ of the length of the optimal Steiner tree, whenever the metric is given by points in the Euclidean plane. Du & Hwang (1990) recently proved this conjecture. Hwang (1976) showed that for rectilinear distances in the plane, this ratio is always at most 3/2. When analyzed in these restricted metrics, Zelikovsky's algorithm can be shown to improve on known results in both cases. Du, Zhang, & Feng (1991) showed that this algorithm is a ρ-approximation algorithm with $\rho < 2/\sqrt{3}$ for Euclidean case, but did not give an explicit value for ρ. Zelikovsky and Berman & Ramaiyer (1992) independently showed that it is an 11/8-approximation algorithm for the rectilinear case. The latter paper also generalized the algorithm to get improved bounds; for example, again using 4-tuples yields a 97/72-approximation algorithm.

Bern & Plassman (1989) showed that the Steiner tree problem is *MAXSNP*-hard in the metric in which each distance is either 1 or 2. As for the TSP, this still leaves open the question about whether there exists a polynomial approximation scheme for the geometrically defined metrics. To define an asymmetric Steiner tree problem, one can consider the setting in which there also is a specified root terminal, and you wish to find a branching in which each terminal is reachable from the root. It is not hard to show that there is a simple approximation preserving reduction from the set covering problem, and hence a polynomial-time algorithm with performance guarantee $o(\log n)$ is not likely to exist (see Section 7).

5. Nasty gaps

The connection between probabilistic proofs and hardness results for approximation questions has greatly aided our understanding of the extent to which near-optimal solutions can be obtained for certain combinatorial optimization problems. However, there are many other problems for which this approach does not yet seem to be relevant. Of particular interest are those problems for which there is a constant ρ-approximation algorithm known, and for some $\rho' < \rho$, it is possible to prove by *elementary methods* that no ρ'-approximation algorithm exists unless *P*=*NP*. In this section, we will discuss a few examples of this kind.

In the problem of *scheduling identical parallel machines subject to precedence constraints*, we are given a collection of jobs $\{1, 2, \dots, n\}$ and m machines, where job j has a processing requirement of p_j time units, and there is a partial order $\prec$ on the jobs, where $j \prec k$ requires that job j complete its processing by the time job k starts. Each job must be assigned to one machine to be processed without interruption for its specified amount of time. Each machine can be assigned at most one job at a time. The aim is to find a schedule for all jobs so as to minimize the maximum job completion time.

One of the earliest results on approximation algorithms deals with this model. Graham (1966) showed that the following is a 2-approximation algorithm: the jobs are listed in any order that is consistent with the precedence constraints, and whenever a machine becomes idle, the next job on the list with all of its predecessors completed is assigned to that machine; if no such job exists, then the machine is left idle until the next machine completes a job. Let $OPT_{MS}(I)$ denote the optimal value for an instance I.

To show that the schedule produced is no more than twice the optimal length, we will partition the schedule into two classes of time intervals; for each, it will be easy to see that its total length is at most $OPT_{MS}(I)$. Construct a chain $\mathcal{C}$ (backwards) in the partial order $\prec$ as follows: start with the job that finishes last in the schedule, select its predecessor in $\prec$ that finishes last, and iterate this process until the new job identified does not have any predecessors in $\prec$. Consider the time during which a job in $\mathcal{C}$ is being processed. Since $\mathcal{C}$ is a chain, the total length of these intervals is at most $OPT_{MS}(I)$. However, at any other time, no machine is idle, since the next job in the chain is already available to be processed, all of its predecessors having been completed. Since the total time in which no machine is idle is at most $OPT_{MS}(I)$, this completes the analysis.

Lenstra & Rinnooy Kan (1978) showed that, even if each job j has processing requirement $p_j = 1$, $j = 1, \dots, n$, deciding if there is a schedule of length 3 is NP-complete. This implies that for any $\rho < 4/3$, no ρ-approximation algorithm exists unless P=NP. However, deciding if there is a schedule of length 2 is quite easy, and hence no stronger bound can be proved via this approach. It would be a considerable advance in the theory of using probabilistic proofs to prove hardness results for approximation problems if it could be used to improve upon the lower bound of 4/3.

Another example of a problem with a "nasty gap" is the following problem of *scheduling unrelated parallel machines*: if job j is assigned to machine i, then it requires p_{ij} units of processing time; each job must be assigned to exactly one machine; the completion time of a machine is the total processing time of the jobs assigned to it. We wish to find an assignment that minimizes the maximum completion time of any machine; we shall call this the *length* of the assignment. The best known approximation algorithm for this problem is due to Lenstra, Shmoys, & Tardos (1990), which relies on a deterministic rounding approach to find solutions within a factor of two of optimal.

We can formulate the problem of deciding if there exists an assignment of length at most T as the following integer program:

$$\sum_{j=1}^{n} p_{ij} x_{ij} \leq T, \qquad \text{for each } i = 1, \dots, m, \tag{13}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad \text{for each } j = 1, \dots, n, \tag{14}$$

$$x_{ij} \in \{0,1\}, \qquad \text{for each } i = 1, \dots, m,\ j = 1, \dots, n. \tag{15}$$

To obtain a linear relaxation, replace the constraint (15) by

$$x_{ij} = 0, \qquad \text{if } p_{ij} > T, \tag{16}$$

$$x_{ij} \geq 0, \qquad \text{for each } i = 1, \ldots, m,\ j = 1, \ldots, n. \tag{17}$$

If the linear relaxation does not have a feasible solution, then clearly no integer solution exists. Otherwise, find an extreme point x^*. Potts (1985) observed that this solution has at most $m+n$ non-zeroes, and hence at most m of the jobs are scheduled fractionally. He used this observation, for a weaker linear relaxation, to obtain a 2-approximation algorithm if the number of machines is a constant.

Lenstra, Shmoys & Tardos (1990) showed that x^* can be rounded to a good assignment. Intuitively, we would like to do this so that each machine is assigned at most one of the fractionally scheduled jobs; since $x^* > 0$ implies that $p_{ij} < T$, this ensures that the length of the rounded assignment is at most $2T$. This linear program is a generalized flow problem, and well-known properties of extreme points for the generalized flow problem can be used to show that such a matching of fractionally assigned jobs and machines must exist. Furthermore, the matching can be found by a simple procedure that runs in $O(m+n)$ time. This procedure can be converted into a 2-approximation algorithm for the optimization version of the problem, by performing a bisection search for the minimum T such that the linear relaxation has a feasible solution.

Lenstra, Shmoys, & Tardos (1990) also proved that for any $\rho < 3/2$, no ρ-approximation algorithm exists unless *P=NP*. In fact, the hardness result holds if, for each job j, its processing time on any machine is either p_j or ∞. It would be nice to show that, perhaps starting with this special case, that there are algorithms with a matching performance guarantee. As for the problem of scheduling on identical parallel machines subject to precedence constraints, it seems difficult to improve the lower bound based on the new randomized characterization of *NP*.

We conclude this section with other deterministic rounding results, for a generalization of the problem of scheduling unrelated parallel machines. In the *generalized assignment problem*, when job j is assigned to machine i, it not only adds p_{ij} to the load on this machine, but it incurs a cost c_{ij} as well. Now we wish to decide if there is an assignment of length T and *total* cost C. Similar rounding procedures for this problem were given independently by Trick (1991) and Lin & Vitter (1992). We shall outline the result of Lin & Vitter (1992), which shows how to compute, for any $\epsilon > 0$, an assignment of total cost $(1+\epsilon)C$ and length $(2+1/\epsilon)T$, assuming that one of total cost C and length T exists.

The integer programming formulation consists of the constraints (13)–(15) plus

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \leq C. \tag{18}$$

We first check if the linear relaxation where constraints (15) are replaced by (16) and (17) has a feasible solution. If not, then no assignment of cost at most C and length at most T exists. Otherwise, let x^* denote a feasible solution. In this fractional assignment, each job j incurs a cost $\bar{c}_j = \sum_{i=1}^{m} c_{ij} x^*_{ij}$, $j = 1, \ldots, n$. Let $\mathcal{C}$ denote the set of pairs (i, j) such that $c_{ij} > (1+\epsilon)\bar{c}_j$; let $w_j = \sum_{(i,j)\notin\mathcal{C}} x^*_{ij}$, the fraction of job j that is assigned relatively cheaply in the fractional assignment. Note that $w_j > \epsilon/(1+\epsilon)$.

Consider a new fractional assignment $\bar{x}$, where

$$\bar{x}_{ij} = \begin{cases} x^*_{ij}/w_j & \text{if } (i,j) \notin \mathcal{C}, \\ 0 & \text{if } (i,j) \in \mathcal{C}. \end{cases}$$

Since we have just zeroed out some components of x^* and rescaled, $\bar{x}$ clearly satisfies (14). Furthermore, since $w_j > \epsilon/(1+\epsilon)$ for each $j = 1, \ldots, n$,

$$\sum_{j=1}^{n} p_{ij}\bar{x}_{ij} \le (1 + 1/\epsilon)T, \qquad \text{for each } i = 1, \ldots, m. \tag{19}$$

Applying the rounding theorem of Lenstra, Shmoys, & Tardos (1990) to $\bar{x}$, we get a solution of total cost at most $\sum_{j=1}^{n}(1+\epsilon)\bar{c}_j \le (1+\epsilon)C$ and length at most $(2 + 1/\epsilon)T$. Lin & Vitter (1992) also give several other applications of this type of rounding technique, which they call filtering.

Shmoys & Tardos (1993) give an improved rounding technique that rounds any feasible solution in polynomial time to an assignment of cost at most C and length at most $2T$. The rounding is done by finding a minimum-cost matching in an associated bipartite graph. The graph is constructed so as to guarantee that any matching has the property that the associated assignment has length at most $2T$; furthermore, the cost of the matching is equal to the total cost of the assignment. The existence of a good rounding is proved by exhibiting a fractional matching of cost C. Thus, by the theorem of Birkhoff and Von Neumann there must also exist an integer matching of cost at most C. The fractional matching is easily derived from the feasible solution to the linear relaxation.

6. Primal-dual algorithms for network design problems

Approximation algorithms are often based on linear programming by designing a primal-dual approximation algorithm: that is, we simultaneously construct an integer primal solution and a dual solution so that their values are provably close to each other. In Section 2, we gave an example of how such an approach can be used for the vertex cover problem. In that case, the primary motivation was to improve the efficiency of the approximation algorithm. However, in many cases the most effective approximation algorithms are based on this approach.

Primal-dual algorithms have been particularly successful for network design problems. In the *minimum-cost survivable network design problem*, we are given an undirected graph $G = (V, E)$ where each edge $e \in E$ has a given cost $c(e) \ge 0$, and for each pair of nodes u and v, there is a specified connectivity requirement $r(u,v)$; the aim is to find a minimum-cost subgraph $G' = (V, E')$ such that there are $r(u,v)$ edge-disjoint paths between each pair $u, v \in V$.

This problem is a common generalization of a number of combinatorial optimization problems. For many of these, the requirements are defined implicitly by node-connectivity requirements: for each $v \in V$, there is a value $r(v)$, and $r(u,v) = \min\{r(u), r(v)\}$. For example, if $r(v) = 1$ for each $v \in V$, then we merely require a minimum-cost connected subgraph; in other words, this is the minimum spanning tree problem. More generally, if $r(v) = k$ for each $v \in V$, then we are looking for the minimum k-edge-connected network. If each $r(v) \in \{0, 1\}$, then this is the minimum Steiner tree problem; a node v with $r(v) = 1$ is a terminal, whereas one with $r(v) = 0$ is a Steiner node. There has been a great deal of research on these and other special cases, and we shall not attempt to survey this literature. We shall focus instead on results from one particular thread of research that has attempted to use a natural linear programming relaxation to obtain good performance guarantees for these problems.

Goemans and Bertsimas (1993) consider the following linear relaxation of the survivable network design problem in the case when $r(u,v) = \min\{r(u), r(v)\}$:

$$\text{minimize} \sum_{e \in E} c(e)x(e) \tag{20}$$

subject to

$$\sum_{e \in \delta(S)} x(e) \geq \max_{(u,v) \in \delta(S)} r(u,v), \qquad \text{for each } S \subset V, S \neq \emptyset, \tag{21}$$

$$x(e) \geq 0, \qquad \text{for each } e \in E. \tag{22}$$

Their main result is to show a strong structural property of this linear program whenever the edge costs satisfy the triangle inequality: $c(u,v) + c(v,w) \geq c(u,w)$ for each $u, v, w \in V$. The property is called the *parsimonious property* and it is as follows: for any subset of nodes D, if we add the constraints

$$\sum_{e \in \delta(u)} x(e) = \max_{v \in V-u} r(u,v), \qquad \text{for each } u \in D, \tag{23}$$

then the optimal value is unchanged. The parsimonious property can be used to show that the ratio between the cost of the Steiner tree produced by the minimum spanning tree heuristic and the linear programming relaxation is less than 2.

Goemans & Bertsimas (1993) also consider the variant of the survivable network design problem in which the network constructed may contain multiple copies of each edge. They give a natural extension of the Steiner tree heuristic for the case when the connectivity requirements are of the form $r(u,v) = \min\{r(u), r(v)\}$. Let $r_{\max}$ denote the value $\max_{u,v \in V} r(u,v)$. The algorithm works in $r_{\max}$ phases, where in phase $i = 1, \ldots, r_{\max}$, the algorithm augments the connectivity on the subset of nodes $V_i = \{v \in V : r(v) \geq i\}$, by running the Steiner tree heuristic with the set of terminals given by V_i. They prove that if there are k distinct values $r(v)$, $v \in V$, then this is a $\min\{2k, 2\mathcal{H}(r_{\max})\}$-approximation algorithm where $\mathcal{H}(k) = 1+1/2+1/3+\cdots+1/k$ is the harmonic function.

The idea of using a primal-dual approach in this context was introduced by Agrawal, Klein, and Ravi (1991), who gave a $2\log r_{\max}$-approximation algorithm for the same problem, but where the connectivity requirements need not be of the restricted form. The algorithm uses scaling techniques to satisfy one bit of the connectivity requirements at a time. The core of the algorithm is a 2-approximation algorithm for the problem with each $r(u,v) \in \{0,1\}$, which can be viewed as a *Steiner forest problem*: there are sets of terminals $T_1, T_2, \ldots, T_p$, and we wish to find the minimum-cost forest such that each set T_j is contained in some connected component of the forest.

Goemans & Williamson (1992) give a very elegant generalization that explicitly relies on a linear programming formulation to give a primal-dual algorithm that is in much the same spirit as primal-dual algorithms for polynomially solvable combinatorial optimization problems, such as the minimum-cost perfect matching problem. They consider problems specified by the integer program

$$\text{minimize} \sum_{e \in E} c(e)x(e) \tag{24}$$

subject to

$$\sum_{e \in \delta(S)} x(e) \geq f(S), \qquad \text{for each } S \subset V, S \neq \emptyset, \tag{25}$$

$$x(e) \in \{0,1\}, \qquad \text{for each } e \in E, \tag{26}$$

for a broad class of $\{0,1\}$-functions f, which they call *proper functions*. A proper function must satisfy two properties: $f(S) = f(V-S)$ for each $S \subseteq V$, and if A and B are disjoint, then $f(A \cup B) \leq \max\{f(A), f(B)\}$. In addition to the $\{0,1\}$-problems discussed above, this framework also captures, for example, the $s-t$ shortest path problem and minimum T-join problem.

In order to explain this algorithm, we first formulate the dual of its linear programming relaxation:

$$\text{maximize} \sum_{S \subset V} f(S)y(S) \tag{27}$$

subject to

$$\sum_{S: e \in \delta(S)} y(S) \leq c(e), \qquad \text{for each } e \in E, \tag{28}$$

$$y(S) \geq 0, \qquad \text{for each } S \subset V, S \neq \emptyset. \tag{29}$$

The primal-dual algorithm maintains the following invariant: there is a partition of the node set $\mathcal{C}$, such that the current primal solution is a forest with components given by $\mathcal{C}$. The initial primal solution is the empty set, so that each part in $\mathcal{C}$ is initially just a single vertex. A feasible dual solution is maintained throughout; initially each $y(S) = 0$. Each iteration of the algorithm selects one edge (u, v) that connects two distinct components, and then merges the two components. The edge is selected in the following way. Each component C of $\mathcal{C}$ for which $f(C) = 1$ is considered *active*; that is, the current primal solution is infeasible with respect to C. To select the new edge to be added, we set $y(C) \leftarrow y(C) + \delta$ for each active component C, where δ is set to be the largest value that maintains dual feasibility. Consequently, this change causes one of the constraints (28) to be satisfied with equality. This tight constraint determines the new edge to be added to the primal solution. When there are no active components, we consider each edge that has been included in the solution in reverse order, and discard each that is not necessary for the remaining solution to be feasible. Some intuition behind this clean-up phase can be gained by considering the algorithm in the case that it is computing a shortest $s - t$ path: the algorithm grows single-source shortest-path trees from s and t until the two trees meet; then each edge which is not on this $s - t$ path is deleted.

Another application of this result is to obtain a faster approximation algorithm for the minimum-cost perfect matching problem for costs that satisfy the triangle inequality. Although this matching problem can be solved in polynomial time, the best known algorithm for it runs in $O(n^3)$ time on dense graphs. In contrast, the algorithm presented above can be implemented to run in $O(n^2 \log n)$ time on dense graphs. To capture the matching problem, consider $f(S)$ which is defined to be 0 if $|S|$ is even, and 1 if $|S|$ is odd. The resulting primal solution will not necessarily be a matching; the algorithm finds a forest in which each component has an even number of nodes. However, each component can be converted into a matching of no greater cost by considering the tour of these nodes found by shortcutting a "double-tree" traversal, and then, as in the analysis of Christofides' algorithm, choosing the cheaper of the two matchings formed by taking alternate edges in the tour.

Goemans & Williamson (1992) also apply their result to the network design problem given by any proper function, where each edge may be included any number of times; this yields a $2\mathcal{H}(r_{\max})$-approximation algorithm in this setting. Aggarwal & Garg (1994) show how to obtain a performance guarantee of $2\mathcal{H}(k)$, where k denotes the number of nodes v for which $f(v)$ is positive.

Although we have seen quite a number of results for the survivable network design problem when multiple copies of each may be included, this problem has fewer applications than the variant in which each edge can be included at most once. It appears to be much harder to obtain good results for this variant, and we shall now focus on describing the results known for it. The first step was due to Klein & Ravi (1993), who gave a 3-approximation algorithm for the network design problem for all proper functions with range $\{0, 2\}$.

Williamson, Goemans, Mihail, and Vazirani (1993) gave the first approximation algorithm for arbitrary proper functions. Their algorithm incrementally builds a feasible solution by adding edges in a series of $r_{\max}$ phases, where each phase is one execution of the {0,1}-algorithm described above. A cut S is *deficient* if the number of edges of $\delta(S)$ that are in the current solution is less than the requirement for that cut, $f(S)$. In each phase, we target the deficient cuts by formulating a {0,1}-problem given by a function $h(S)$ which is 1 exactly when S is deficient. The {0,1}-algorithm delivers a solution that is within a factor of 2 of the overall optimum, and so the algorithm is a $2r_{\max}$-approximation algorithm. Unfortunately, h need not be a proper function, but it can be shown to be an *uncrossable* function, for which Williamson et al. also give a 2-approximation algorithm. In several particular cases, the algorithm can be shown to have a stronger performance guarantee; for example, if $f(S) \in \{0, 1, 2\}$ for all S, then it is a 3-approximation algorithm. Gabow, Goemans, and Williamson (1993) give a more efficient implementation of this approach.

It turns out that the algorithm of Williamson et al. can be improved in a rather simple way. Let the *deficiency* of a cut S be the number edges of $\delta(S)$ that must still be added to the current solution to bring the number of edges to be at least $f(S)$. Let Δ be the maximum deficiency of a cut. Goemans, Goldberg, Plotkin, Shmoys, Tardos, and Williamson (1994) propose that in each phase, only the cuts with deficiency Δ be targeted; that is, $h(S)$ is set to 1 for only those cuts. By arguing that the optimum value for this {0,1}-problem is at most the overall optimum divided by Δ, they show that this yields a $2\mathcal{H}(r_{\max})$-approximation algorithm. It remains an interesting question whether there exists an approximation algorithm with a constant performance guarantee for this problem.

Khuller & Vishkin (1992) give a simple 2-approximation algorithm for the case in which $r(v) = k$ for each $v \in V$ (that is, the minimum-cost k-connected subgraph). The idea of the algorithm is quite simple; replace each undirected edge (u, v) by two directed edges, (u, v) and (v, u), each of cost equal to the cost of the undirected edge. Choose an arbitrary root node v and find k disjoint branchings into v of minimum total cost, as well as a minimum-cost family of k disjoint branchings out of v. Include an (undirected) edge in the solution if either directed copy is used in either solution. It is not hard to see that this is a feasible solution, and since the cost of either family of branchings is a lower bound on the optimum, this is a 2-approximation algorithm. A natural first step towards a constant performance guarantee for the network design problem specified by proper functions would be to consider the Steiner version of the k-connected subgraph problem, that is, $r(v) \in \{0, k\}$ for each $v \in V$.

Hochbaum & Naor (unpublished manuscript) consider another special case, in which the proper function $f(S)$ has the property that $|S| > p \Rightarrow f(S) = 0$, and give an $O(\log p)$-approximation algorithm for this case.

7. Logarithmic performance guarantees

For quite a number of *NP*-hard combinatorial optimization problems, no approximation algorithm is known that has a constant performance guarantee. The next best alternative is to give an approximation algorithm in which the performance guarantee is a slowly growing function of the input size. One natural class of guarantees is when the problem can be solved within a logarithmic or polylogarithmic factor of optimal. In this section, we shall present several results of this type.

In the *minimum-cost set covering problem*, we are given a family of sets $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$ where $S_i \subseteq \{1, \ldots, n\}$, $i = 1, \ldots, m$, and non-negative costs c_i, $i = 1, \ldots, m$, and we wish to select a subset $\mathcal{F}' \subseteq \mathcal{F}$ such that $\cup_{S \in \mathcal{F}'} S = \{1, \ldots, n\}$, so as to minimize the total cost of the sets in $\mathcal{F}'$. Consider the following greedy algorithm: we iteratively build a set cover by selecting an additional set in each iteration until the union includes all elements $\{1, \ldots, n\}$; in each iteration, we compute the *marginal cost* of covering a new element for each set: the ratio of its cost to the number of its elements that were not in the union of the sets already selected; we choose the set S_i for which the marginal cost is smallest, and include it in the cover.

Johnson (1974a) and Lovász (1975) independently showed that the greedy algorithm is an $\mathcal{H}(n)$-approximation algorithm for the case in which each cost c_i is 1, where the harmonic function $\mathcal{H}(n)$ is at most $1 + \ln n$. Chvátal (1979) showed that it also is an $\mathcal{H}(n)$-approximation algorithm in the case of arbitrary non-negative costs. In fact, the greedy algorithm can be viewed as a primal-dual algorithm. If one considers the natural linear programming relaxation of this problem,

$$\text{minimize} \sum_{i=1}^{m} c_i x_i \tag{30}$$

subject to

$$\begin{aligned} \sum_{i: j \in S_i} x_i &\geq 1, && \text{for each } j = 1, \ldots, n, && (31) \\ x_i &\geq 0, && \text{for each } i = 1, \ldots, m, && (32) \end{aligned}$$

then its dual is

$$\text{maximize} \sum_{j=1}^{n} y_j \tag{33}$$

subject to

$$\begin{aligned} \sum_{j \in S_i} y_j &\leq c_i, && \text{for each } i = 1, \ldots, m, && (34) \\ y_j &\geq 0, && \text{for each } j = 1, \ldots, n. && (35) \end{aligned}$$

As the greedy algorithm constructs an integer primal solution, we can view it as computing a dual solution as well: if the element j is first covered in the iteration in which S_i is selected and the associated marginal cost is c_i/k, then set $y_j = c_i/k$. It is clear that by setting the dual variables in this way, the dual objective function value is exactly equal to the cost of the set cover computed. However, this dual solution is not feasible. The crux of the performance guarantee is that if we set y_j instead to $c_i/(k\mathcal{H}(n))$, then this is a feasible dual solution.

Lund & Yannakakis (1993) have given the first significant evidence that no approximation algorithm for the unweighted set cover problem can perform substantially better than the greedy algorithm. Using a result of Feige & Lovász (1992), they showed that if there exists a $\rho \log_2 n$-approximation algorithm with $\rho < .25$, then $NP \subseteq DTIME(n^{\text{poly} \log n})$. (In comparison, the constant for the greedy algorithm is roughly $\rho = .7$.) Bellare, Goldwasser, Lund, & Russell (1993) showed that non-approximability results for the unweighted set cover problem can be obtained based on weaker complexity assumptions: they showed that no constant performance guarantee exists unless P=NP, and they showed that, unless $NP \subseteq DTIME(n^{\log \log n})$, there does not exist a $\rho \log_2 n$-approximation algorithm for any $\rho < .125$. There are several problems known to be equivalent to the set covering problem, in terms of the performance guarantees that can be obtained. Most interesting among these is the minimum dominating set problem.

One of the most important recent breakthroughs in obtaining approximation algorithms with a (poly)logarithmic performance guarantee is for the problem of finding balanced cuts in graphs. Given an undirected graph $G = (V, E)$, a cut $S \subseteq V$ is *β-balanced* if $\min\{|S|, |\bar{S}|\} \geq \beta|V|$, where $\bar{S}$ denotes $V - S$. If we are given G along with a nonnegative cost $c(e)$ for each $e \in E$, the minimum-cost β-balanced cut is to find a β-balanced cut S so as to minimize $\sum_{e\in\delta(S)} c(e)$. The seminal paper of Leighton & Rao (1988) showed how to obtain, for any constants $\beta \leq 1/3$ and $\epsilon > 0$, a β-balanced cut that has cost $O(\log n)$ times the cost of an optimal $(\beta + \epsilon)$-balanced cut. Leighton & Rao also demonstrated that this is a fundamental primitive in the design of approximation algorithms, by giving a broad range of applications of this algorithm to a variety of other optimization problems.

The approximation algorithm of Leighton & Rao that finds balanced cuts is based on an algorithm to find good *quotient cuts*. More precisely, we instead focus on finding a set $S \subseteq V$, so as to minimize $\sum_{e\in\delta(S)} c(e)/\min\{|S|, |\bar{S}|\}$. Observe that if we have any algorithm to find a "good" cut in a graph, we can use this repeatedly to find a 1/3-balanced cut S as follows. In iteration i, find a cut S_i, and add the smaller of S_i and $\bar{S}_i$ to the current cut S, thereby deleting these vertices from the remaining graph. Since the smaller of S_i or $\bar{S}_i$ always has at most half the vertices, we can be sure that at some point the current cut S has between $|V|/3$ and $2|V|/3$ vertices; that is, it is 1/3-balanced; when this happens, the algorithm terminates. By trying to find a minimum quotient cut in each iteration, we are adopting a greedy strategy in much the same way as we found good solutions for the set covering problem. Unfortunately, finding the minimum quotient cut is an *NP*-hard problem, but finding an $O(\log n)$-approximation algorithm for it suffices to prove the theorem about balanced cuts given above.

In fact, Leighton & Rao actually focus on another cut problem, in which the objective is to find a cut that minimizes $\sum_{e\in\delta(S)} c(e)/\{|S||\bar{S}|\}$. As far as approximation algorithms are concerned, this new problem is asymptotically equivalent to the quotient cut problem, since the $\max\{|S|,|\bar{S}|\}$ varies by less than a factor of 2 over all cuts S. The advantage of dealing with this new objective is that it is the combinatorial dual of a multicommodity flow problem, where we wish to send one unit of flow between each pair of vertices subject to joint capacity constraints $c(e)$ for each $e \in E$, and it is this connection that drives their approximation algorithm.

Before giving a more detailed discussion of the way in which finding near-optimal balanced cuts and multicommodity flows are related, we shall discuss a couple of the applications of finding well-balanced cuts. One of the principle motivations was to give efficient simulations of PRAMs on arbitrary networks. The fact that the performance guarantee of the approximation algorithm is logarithmic is crucial in this setting, since this makes it possible for the simulation's running time to remain polylogarithmic. We shall, however, focus instead on applications to specific combinatorial optimization problems.

In the *minimum cut linear arrangement problem*, we are given a graph G, and the aim is to index the n vertices $\{1,2,\ldots,n\}$ so as to minimize the maximum of $|\delta(S)|$ for all cuts S of the form $\{1,2,\ldots,j\}$. Let $OPT_{LA}(G)$ denote the value of the optimal solution for a given graph G. The following is an $O(\log^2 n)$-approximation algorithm for this problem, that works by divide-and-conquer: find a (good) 1/3-balanced cut S, order all of the vertices of S before the vertices of $\bar{S}$, and recurse on each of S and $\bar{S}$ to compute the rest of the order. Observe that G has a 1/2-balanced cut of value at most $OPT_{LA}(G)$: take the optimal solution to the minimum cut linear arrangement problem, and consider the cut given by $S = \{1,2,\ldots,\lfloor n/2\rfloor\}$. Hence the initial call produces a cut with $O(\log n)OPT_{LA}(G)$ edges. If we consider any cut $S=\{1,2,\ldots,j\}$ with respect to the final order computed, each edge in $\delta(S)$ is contributed by some level ℓ of the recursion, and the number of edges from each level ℓ is $O(\log n)OPT_{LA}(G)$. The balancing condition ensures that there are $O(\log n)$ levels of recursion.

Similar divide-and-conquer strategies drive each of the applications of finding balanced cuts to derive approximation algorithms. Another nice application given by Leighton & Rao is to obtain an $O(\log^2 n)$-approximation algorithm for the *minimum-cost feedback arc set problem*: given a directed graph $G=(V,A)$, where each arc $a\in A$ has an associated nonnegative cost $c(a)$, find a set of arcs $F\subseteq A$ such that the graph $G=(V,A-F)$ is acyclic, so as to minimize the total cost of the arcs in F, $\sum_{a\in F} c(a)$. The analogous directed cut is to find arcs to delete such that the resulting graph is divided into well-balanced strongly connected components, which might, at first glance, appear less natural. Klein, Agrawal, Ravi & Rao (1990) later gave other applications, most notably, for the problem of computing good pivoting strategies in Gaussian elimination to minimize fill-in.

In the remainder of this section, we wish to briefly sketch the ideas behind these results, and in particular explore the connection between such cut problems and multicommodity flow. For illustrative purposes, we focus on a closely related cut problem, which we shall call the *minimum-cost multicut problem.* We are given a graph $G = (V, E)$ where each edge $e \in E$ has a non-negative cost $c(e)$, along with k pairs of nodes $(s_1, t_1), \ldots, (s_k, t_k)$; we wish to partition the vertices into any number of parts, $V_1, \ldots, V_l$, such that s_i and t_i are in different parts, for each $i = 1, \ldots, k$, so as to minimize the total cost of edges with endpoints in different parts. If $k = 1$, this is the standard minimum cut problem, for undirected graphs. A combinatorially dual problem is the *maximum multicommodity flow problem*: given the same input, find a flow f_i between s_i and t_i of value ν_i such that, for each edge e, the total flow on e, $\sum_{i=1}^{k} f_i(e)$, is at most $c(e)$, so as to maximize the total flow value, $\sum_{i=1}^{k} \nu_i$. Once again, if $k = 1$, this is the standard maximum flow problem. Of course, in this case, there is the celebrated max-flow min-cut theorem of Ford & Fulkerson (1956), that states that the value of the maximum flow is equal to the value of the minimum-cost cut. This theorem does not generalize to larger k. However, the weak duality relationship still holds: if there is a multicut of cost α, then clearly any multicommodity flow has value at most α.

The minimum-cost multicut problem is *NP*-hard. The maximum multicommodity flow problem is a linear program, and hence can be solved in polynomial time. Note, however, that unlike the single commodity flow problem, there need not exist an optimal solution to this linear program that is integral. The linear programming dual of the maximum multicommodity flow problem can be explained as follows. Imagine the input graph G as a system of pipes, corresponding to the edges, where the cross section of the pipe corresponding to edge e has area equal to $c(e)$. For each edge e, there is a nonnegative dual variable $\ell(e)$ which can be viewed as the length of the corresponding pipe. Thus, the total volume W of the pipe system is $\sum_{e \in E} c(e)\ell(e)$. Suppose that there is a multicommodity flow of total value ν. Let d_i denote the length of the shortest path (with respect to ℓ) between the source s_i and its corresponding sink t_i; then each unit of flow between s_i and t_i must use a total volume of d_i as it travels. Hence, if we specify the length function in units such that $\min_i d_i = 1$, then it is clear that the value of a maximum multicommodity flow ν^* is at most the total volume W. The dual linear program is to compute the length function ℓ (scaled such that $\min_i d_i = 1$) so that the total volume of the system is minimized. By the duality theorem of linear programming, the value ν^* of the maximum multicommodity flow is equal to the minimum volume W^*.

Garg, Vazirani, & Yannakakis (1993) give an algorithm that takes as input a pipe system of volume W, and computes a multicut of cost $O(\log k)W$. Since we can compute a pipe system of the optimal volume W^* in polynomial time, we can then obtain a multicut of cost $O(\log k)W^* = O(\log k)\nu^*$. However, since ν^* is a lower bound on the cost of the minimum multicut, we see that this yields an $O(\log k)$-approximation algorithm for the minimum-cost multicut problem.

We next sketch the procedure to compute a multicut, given a pipe system of volume W with a particular length function ℓ. In iteration l, the algorithm computes a subset of vertices V_l such that V_l does not contain both s_i and t_i, for each $i = 1, \ldots, k$, and for at least one of these pairs, s_j and t_j, contains exactly one of s_j and t_j. These vertices are deleted, and this yields a smaller pipe system induced by the remaining vertices. The algorithm continues to partition the remaining vertices until all source-sink pairs are separated by the current multicut. In each iteration, choose some s_i and t_i that are still unseparated. Consider a sphere of some fixed radius r around s_i in the pipe system: more precisely, take that part of the pipe system that is within distance r of s_i with respect to ℓ. The part of the multicut V_l chosen in this iteration is the set of vertices that are inside the sphere for a judicious choice of radius r. Imagine adding a point "volume" of value W/k at s_i; then r is set to the smallest value such that $2\ln(2k)$ times the volume inside the sphere is greater than the total cost of the cut defined. A straightforward computation shows that $r < 1/2$, and hence no source-sink pair is contained within the sphere: each such pair is at least distance 1 apart. Thus, the algorithm eventually produces a multicut. This calculation involves the solution of a differential equation: observe that the value of the cut defined by the sphere of radius r is the derivative of the volume contained as a function of r. Finally, the total cost of the multicut found can be bounded iteration by iteration: the edges added to the multicut in each iteration have cost at most $2\ln(2k)$ times the volume of the sphere. The total volume of the pipe system is at most W, and the total of the "point" volumes added is at most W, and so the total cost of the multicut is at most $4\ln(2k)W$.

Since the publication of the paper by Leighton & Rao (1988), there has been a great deal of work in extending their work to more general settings. The reader is referred to the survey of Tardos (1993) for an annotated bibliography of these results. Very recently, Chung & Yau (1994) claim to have obtained a polynomial-time algorithm based on different techniques that produces a 1/3-balanced cut of cost within a constant factor of the optimal $(1/3+\epsilon)$-balanced cut. It remains an interesting problem to derive true approximation algorithms for these problems, in the sense that the cut produced has cost close to the optimal cut with the same balance restriction. Furthermore, for the various cut problems discussed above, as well as the applications of them, no non-trivial lower bounds are known for the performance guarantees that can be obtained.

8. Approximations Algorithms with Small Absolute Error Bounds

In the preceding sections, we have primarily focused on the design and analysis of ρ-approximation algorithms. For some problems, this is not the most appropriate type of performance guarantee.

For example, consider the *bin-packing problem*: we are given n pieces with specified sizes $a_1, a_2, \ldots, a_n$, such that $1 > a_1 \geq a_2 \geq \cdots \geq a_n > 0$; we wish to pack the pieces into bins, where each bin can hold any subset of pieces of total size at most 1, so as to minimize the number of bins used. To decide if two bins suffice is exactly the partition problem, and hence is *NP*-complete. Thus, there cannot exist a ρ-approximation algorithm for any $\rho < 3/2$ unless *P*=*NP*. However, consider the First-Fit-Decreasing algorithm, where the pieces are packed in order of decreasing size, and the next piece is always packed into the first bin in which it fits. If $FFD(I)$ denotes the number of bins used by this algorithm on input I, and $OPT_{BP}(I)$ denotes the number of bins used in the optimal packing, then Johnson (1974a) proved that $FFD(I) \leq (11/9)OPT_{BP}(I) + 4$ for any input I. Yue (1991) has subsequently claimed that $FFD(I) \leq (11/9)OPT_{BP}(I) + 1$, which, incidentally, implies that it is a 3/2-approximation algorithm.

Thus, significantly stronger results can be obtained by relaxing the notion of the performance guarantee to allow for small additive terms. In fact, it is completely consistent with our current understanding of complexity theory that there is an algorithm that always produces a packing with at most $OPT_{BP}(I)+1$ bins.

Why is it that we have bothered to mention hardness results of the form, "there does not exist a ρ-approximation algorithm unless *P*=*NP*" if such a result can be so easily circumvented? The reason is that for all of the weighted problems that we have discussed, any distinction between the two types of guarantees disappears; any algorithm guaranteed to produce a solution of value at most $\rho OPT + c$ can be converted to a ρ-approximation algorithm. Each of these problems has a natural rescaling property: for any input I and any value κ, we can construct an essentially identical instance I' such that the objective function value of any feasible solution is rescaled by κ. Such rescaling makes it possible to blunt the effect of any small additive term $c < \kappa$ in the guarantee, and make it effectively 0. Observe that the bin-packing problem does not have this rescaling property; there is no obvious way to "multiply" the instance in a way that does not blur the combinatorial structure of the original input. (Think about what happens if you construct a new input that contains two copies of each piece of I!)

Thus, whenever we consider designing approximation algorithms for a new combinatorial optimization problem it is important to consider first whether the problem does have the rescaling property, since that will indicate what sort of performance guarantee to hope for.

There are nice examples of *NP*-hard optimization problems for which approximation algorithms are known that always deliver solutions of value within 1 of optimal: the edge coloring problem and the minimum max-degree spanning tree problem. In the *edge coloring problem* we are given an undirected graph $G = (V, E)$, and we wish to assign a color to each edge of G so that no two edges with a common endpoint are assigned the same color, so as to minimize the number of colors used.

Vizing (1964) showed how to color each graph G with $\Delta(G)+1$ colors, where $\Delta(G)$ is the maximum number of edges incident to a vertex. Since $\Delta(G)$ colors are certainly needed, this algorithm never uses more than one additional color beyond what is optimal. Much later, Holyer (1981) showed that this problem is *NP*-hard; in fact, he showed that recognizing whether a graph can be edge-colored with three colors is *NP*-hard, and hence no ρ-approximation algorithm exists for any $\rho < 4/3$, unless *P*=*NP*.

There is still an interesting open question concerning edge coloring: does there exist an analogous algorithm for multigraphs? It is possible that linear programming can provide an analogue of the result of Vizing (1964). Let $\mathcal{M}$ denote the set of all matchings in a given multigraph $G = (V, E)$, and consider the following linear relaxation of the edge coloring problem:

$$\text{minimize} \sum_{M \in \mathcal{M}} x(M) \tag{36}$$

subject to

$$\sum_{\substack{M \in \mathcal{M}: \\ e \in M}} x(M) \geq 1, \quad \text{for each } e \in E, \tag{37}$$

$$x(M) \geq 0, \quad \text{for each } M \in \mathcal{M}. \tag{38}$$

Seymour has conjectured that the optimal value of this linear program is at most 1 off from the integer optimum. Of course, even if his conjecture were true, we are asking for something stronger: we would like a polynomial-time algorithm to find such a coloring. Incidentally, even though this linear program has an exponential number of variables it can still be solved by the ellipsoid algorithm, since the separation algorithm required to solve its dual is just a maximum-cost matching algorithm.

In the *minimum max-degree spanning tree problem*, we are given a graph G, and wish to find a spanning tree T of G such that $\Delta(T)$ is minimized. Since deciding if there is a spanning tree of maximum degree 2 is just the Hamiltonian path problem, there does not exist a ρ-approximation algorithm for this problem for any $\rho < 3/2$, unless *P*=*NP*. However, Fürer and Raghavachari (1992) gave an elegant algorithm that always produces a spanning tree of maximum degree within 1 of optimal.

We return now to the bin-packing problem. The best known approximation algorithm, due to Karmarkar & Karp (1982), who build on work by Fernandez de la Vega & Lueker (1981), delivers a solution that uses $OPT_{BP}(I) + O(\log^2(OPT_{BP}(I)))$ bins. This is a rather surprising result, and relies on linear programming in an interesting way.

Suppose that we group pieces of identical size, so that there are b_1 pieces of the largest size s_1, b_2 of the second largest size s_2, ..., b_m pieces of the smallest size s_m. We shall describe an integer programming formulation due to Eisemann (1957). Consider the ways in which a single bin can be packed. One of these packings can be described by an m-tuple $(t_1, \ldots, t_m)$, where t_i indicates the number of pieces of size s_i included. We shall call such an m-tuple a *configuration* if $\sum_i t_i s_i \leq 1$. There might be an exponential number of configurations. For example, if $s_m \geq 1/k$, then the number of configurations can be upper-bounded by $\binom{m+k}{k}$. Let N denote the number of configurations, and let $T_1, \ldots, T_N$ be a complete enumeration of them, where t_{ij} denotes the jth component of T_i. This suggests the following formulation:

$$\text{minimize} \sum_{j=1}^{N} x_j \tag{39}$$

subject to

$$\sum_{j=1}^{N} t_{ij} x_j \geq b_i, \quad \text{for each } i = 1, \ldots, m, \tag{40}$$

$$x_j \in \mathbf{N}, \quad \text{for each } j = 1, \ldots, N. \tag{41}$$

This formulation was introduced in the context of designing practical algorithms to find optimal solutions to certain bin-packing problems.

We shall first present an algorithm that, for any given $\epsilon > 0$, finds a packing with at most $(1+\epsilon)OPT_{BP}(I) + O(p(1/\epsilon))$ bins, where p is some polynomial function. Suppose that we find an optimal extreme point of the linear relaxation of this formulation. It must have at most m non-zero components, and so if we simply round up this solution, then we have obtained an integer solution of size at most $OPT_{BP}(I) + m$. Since m is the number of distinct piece sizes, the name of the game in using this formulation is to first round the instance so as to reduce m.

To implement this approach, we must also be able to solve the linear program. Fernandez de la Vega & Lueker (1981) observed that it is easy to ignore small pieces of size at most $\epsilon/2$, since they can be added later to the packing in a way that certainly does not introduce too much error. This implies that there are $O(m^{2/\epsilon})$ variables; hence the linear program for the remaining pieces can be solved in polynomial time.

Fernandez de la Vega & Lueker (1981) present a linear grouping scheme to reduce the number of distinct piece sizes. This scheme works as follows, and is based on a parameter k, which will be set later. Group the pieces of the given input I as follows: the first group consists of the k largest pieces, the next group consists of the next k largest pieces, and so on, until all pieces have been placed in a group. The last group contains h pieces, where $h \leq k$. The rounded instance I' is constructed by discarding the first group, and for each other group, rounding the size of its pieces up to the size of its largest piece. It is easy to see that

$$OPT_{BP}(I') \leq OPT_{BP}(I) \leq OPT_{BP}(I') + k;$$

significantly, we can quickly transform any packing of I' into a packing of I using at most k additional bins (for the discarded pieces). The number of distinct piece sizes is less than n/k, where n is the number of pieces; if we set $k = \lceil \epsilon SIZE(I) \rceil$ where $SIZE(I) = \sum_i a_i \leq OPT_{BP}(I)$, then we see that $n/k \leq 2/\epsilon^2$. Consequently, if we group the pieces in this way, solve the linear program, and then round up this solution, we obtain a packing that uses at most $(1+\epsilon)OPT_{BP}(I) + 2/\epsilon^2 + 1$ bins. Furthermore, the number of constraints and variables of the linear program is now just a constant (that depends exponentially on $1/\epsilon$).

Karmarkar & Karp (1982) improve both the running time of this general approach, and its performance guarantee. The more technical improvement is the former, where they show how the ellipsoid algorithm can be used to approximately solve this linear program within an additive error of 1 in time bounded by a polynomial in m and $\log(n/a_n)$. This implies that the approximation scheme described above can now be implemented to run in time bounded by a polynomial in $1/\epsilon$ and the size of the input. More significantly, they give a more sophisticated grouping scheme that implies much stronger performance guarantees. We will show how to use this scheme to obtain algorithm that finds a packing with $OPT(I) + O(\log^2(OPT(I)))$ bins. Note that we can assume, without loss of generality, that $a_n \geq 1/SIZE(I)$, since smaller pieces can again be packed later without changing the order of magnitude of the absolute error.

The geometric grouping scheme works as follows: process the pieces in order of decreasing size; close the current group whenever its total size is at least 2, and then start a new group with the next piece. Let r denote the number of groups, let G_i denote the ith group, and let n_i denote the number of pieces in G_i. Observe that for $i = 2, \ldots, r-1$, we have that $n_i \geq n_{i-1}$. As before, from a given input I we form a new instance I', where a number of pieces are discarded and packed separately. For each $i = 2, 3, \ldots, r-1$, we put n_{i-1} pieces in I' of size equal to the largest piece in G_i. We discard G_1, G_r, and the $n_i - n_{i-1}$ smallest pieces in G_i, $i = 2, \ldots, r-1$.

First of all, it should be clear that any packing of the rounded instance I' can be used to pack those pieces of the original instance that were not discarded. We will sketch the proof of two further properties of this scheme: the number of distinct piece sizes in I' is at most $SIZE(I)/2$; and the total size of all discarded pieces is $O(\log SIZE(I))$. The first is easy: each distinct piece size in I' corresponds to one of the groups $G_2, \ldots, G_{r-1}$; each of these groups has size at least 2, and so there are at most $SIZE(I)/2$ of them. To prove the second property, suppose, for the moment, that each group G_i has at most one more piece than the previous group G_{i-1}. To bound the total size of the discarded pieces, the total size of each group is at most 3, and so the total size of G_1 and G_r is at most 6. Furthermore, the size of the smallest piece in group G_i is at most $3/n_i$. Since we discard a piece from G_i, $i = 2, \ldots, r-1$, only when G_i is the first group that has n_i pieces and we discard its smallest piece, the total size of these discarded pieces is at most $\sum_{j=1}^{n_r} 3/j$. However, since each piece has size at least $1/SIZE(I)$, $n_r \leq 3SIZE(I)$, and so the total size of the discarded pieces is $O(\log SIZE(I))$. An amortized version of this argument can be used to show that the same bound holds even if the group sizes are not so well behaved.

The geometric grouping scheme can be used to design an approximation algorithm that always finds a packing with $OPT_{BP}(I) + O(\log^2(OPT_{BP}(I)))$ bins. This algorithm uses the geometric scheme recursively. The algorithm applies the grouping scheme, packs the discarded pieces using the First-Fit-Decreasing algorithm (or virtually any other simple algorithm), solves the linear program for the rounded instance, and rounds this solution down to obtain a packing of a subset of the pieces. This leaves some pieces unpacked, and these are handled by a recursive call until the total size of the remaining pieces is less than a specified constant.

Let I denote the original instance, let I' denote the instance on which the first linear program is solved, and let I_1 and I_2 denote the pieces packed based on the integer part of the fractional solution and those left over for the recursive call, respectively. The key to the analysis of this algorithm is that

$$LP(I_1) + LP(I_2) \leq LP(I') \leq LP(I) \leq OPT_{BP}(I).$$

This implies that the only error introduced in each level of recursion is caused by the discarded pieces. Since I_2 is the leftover corresponding to the fractional part of the optimal solution, its total size is at most the number of constraints in the linear program, which is the number of distinct piece sizes in I'; this is at most $SIZE(I)/2$. Hence the size of the instance decreases by a factor of 2 in each level of recursion, and so there are $O(\log SIZE(I))$ levels. In each of these levels, we use $O(\log SIZE(I))$ bins to pack the discarded pieces, and so we obtain the claimed bound.

It would be very interesting to show that the bin-packing problem cannot be approximated within an additive error of 1 (assuming that $P \neq NP$), and the recent progress gives new hope that such a modest goal might be achieved. On the other hand, it seems entirely possible that there exists an algorithm with constant absolute deviation from optimal. In fact, even without any rounding, it is nontrivial to construct instances for which the optimal value of the linear programming formulation differs from the optimal integer value by more than 1, and no result replacing the 1 by a 2 is currently known.

9. Beyond Approximation

In the quarter century that performance guarantees for approximation algorithms were actively studied, no two problems more completely typified our frustration in making substantial progress than the maximum clique problem and the minimum vertex coloring problem. And for no two problems have the implications of the theorem that *NP*=*PCP*$(\log n, 1)$ been more dramatic.

For the maximum clique problem, Johnson (1974b) proposed a simple greedy heuristic. A clique K is constructed by repeatedly placing a vertex v of maximum degree in K, and deleting v and each vertex not adjacent to v, until the entire graph is deleted. Johnson (1974c) showed that if the graph can be vertex colored with k colors, then it delivers an independent set of size $O(\log_k n)$, where n denotes the number of vertices in the input graph. Boppana & Halldórsson (1992) gave an $O(n/\log^2 n)$-approximation algorithm, and this is the best result known to date. Let $OPT_K(G)$ denote the size of the maximum clique in G.

Although the maximum clique problem is purely combinatorial, it does have a rescaling property. For any graph G and constant κ, we can construct the graph G' by taking κ disjoint copies of G and then adding an edge between each pair of vertices that arise from different copies of G. Clearly, $OPT_K(G') = \kappa OPT_K(G)$, and for any clique of size k in G', it induces a clique of size k/κ in one of the original copies. This implies that there is no need to consider asymptotic performance guarantees for the maximum clique problem that introduce lower order error terms.

Garey & Johnson (1976) developed a graph composition technique to show that if there exists a ρ-approximation algorithm for the maximum clique problem for some constant ρ, then there exists a polynomial approximation scheme. Let G^2 be the graph formed by composing a graph G with itself, in the sense that each node of G is replaced by a copy of G, and there is an edge between any pair of nodes in copies of G that correspond to adjacent nodes in G. Note that $OPT_K(G^2) = OPT_K(G)^2$, and that given a clique of size k^2 in G^2 one can efficiently find a clique of size k in G. By applying a ρ^2-approximation algorithm to G^2, we get a set of size $OPT_K(G)^2/\rho^2$, which can then be used to find a clique in G of size $OPT_K(G)/\rho$. By repeatedly applying this technique, we get the claimed result.

This was the state of the art prior to the work of Feige, Goldwasser, Lovász, Safra, & Szegedy (1991), who scaled down the characterization of *NEXP* of Babai, Fortnow, & Lund (1991) to show that the maximum clique problem does not have a 2-approximation algorithm (and hence a polynomial approximation scheme) unless $NP \subseteq DTIME(n^{O(\log\log n)})$. Arora & Safra (1992) strengthened this theorem to rely on the assumption $P \neq NP$. Arora, Lund, Motwani, Sudan, & Szegedy (1992) showed that there exists an $\epsilon > 0$ such that no n^ϵ-approximation algorithm exists unless *P*=*NP*.

Although we will not give the details, we will at least indicate the connection between holographic proofs and the clique problem. As in Section 1, consider a probabilistic verifier for an *NP*-complete language L. Build a graph as follows: for each of the 2^r outcomes of the coin tosses and each of the 2^k possible values of the k holographic proof positions examined, check if this combination leads the verifier to accept, and if so, construct a vertex corresponding to this pair; two vertices are adjacent if the associated values of the proof are consistent (i.e., if the two random strings cause the same bit of the proof to be examined, then the assumed value of that bit must be the same for both vertices). If the input $x \in L$, then the existence of a good holographic proof implies that there is a clique K of size 2^r: for each outcome σ for the coin tosses, let the vertex corresponding to σ along with the correct bits of the holographic proof be in K. On the other hand, we will show that if there is a clique K of size larger than 2^{r-1}, then there exists a holographic proof for which at least 2^{r-1} coin tosses cause the verifier to accept. By the correctness of the verifier, that implies that $x \in L$. We can construct this holographic proof unambiguously by extracting the values as specified by the description of the vertices in K. (Some bits of the proof may be unspecified by this, but they can be chosen arbitrarily). Since the coin tosses corresponding to vertices in K must all be distinct, there are at least $|K|$ coin tosses that cause the verifier to accept. If there is a 2-approximation algorithm for the clique problem, we could distinguish between the cases when the maximum clique size is at least 2^r, and when it is at most 2^{r-1}, and hence decide whether $x \in L$ in polynomial time, which implies that *P*=*NP*.

It is interesting to note that Berman & Schnitger (1992) observed that the existence of an n^ϵ-approximation algorithm for the clique problem, for some constant $\epsilon > 0$, would yield a randomized polynomial approximation scheme for each problem in *MAX SNP*. Alon, Feige, Wigderson, & Zuckerman (unpublished manuscript) have recently derandomized this construction; consequently, the impossibility result of Arora, Lund, Motwani, Sudan, & Szegedy for the maximum clique problem could be derived directly from their result on the maximum satisfiability problem.

The state of the art for the vertex coloring problem prior to 1991 had not been much better than for the maximum clique problem. For the vertex coloring problem, a corollary of the *NP*-completeness of 3-colorability is that no ρ-approximation algorithm can exist with $\rho < 4/3$. The minimum vertex coloring problem can also be shown to have a rescaling property by applying simple graph composition techniques. Garey & Johnson (1976) use a more intricate composition technique to prove that an asymptotic performance guarantee less than 2 would imply that *P*=*NP*.

Johnson (1974c) gave the first nontrivial performance guarantee, by considering the following algorithm: until all vertices are deleted, repeatedly find an independent set (using the analogue of greedy algorithm given above for the complementary problem, the maximum clique problem), color it with a new color, and delete it from the graph. The bound for the independent set algorithm implies that $O(n/\log n)OPT_C(G)$ colors are used, where $OPT_C(G)$ denotes the optimal number of colors for G.

Wigderson (1983) observed that a graph G with $OPT_C(G) = 3$ can be colored with $O(\sqrt{n})$ colors by the following simple algorithm: while the maximum degree node v has degree at least $\sqrt{n}$, color the (bipartite) neighborhood of v with 2 (unused) colors, delete the colored nodes and repeat; color the remaining graph with $\sqrt{n}$ additional colors by repeatedly coloring a node with some color not used at one of its neighbors. Since the first phase can proceed for only $\sqrt{n}$ iterations, only $3\sqrt{n}$ colors are used in total. By using the same idea to recursively color a k-colorable graph G, i.e., applying the approximation procedure for $(k-1)$-colorable graphs until the maximum degree is sufficiently small, Wigderson improved Johnson's bound to $O(n(\log\log n)^2/(\log n)^2)OPT_C(G)$. There has been little subsequent improvement: after considerable effort, the best performance guarantee currently known is only $O(n(\log\log n)^2/(\log n)^3)$ (Halldórsson, 1993).

For 3-colorable graphs, Blum (1991) improved Wigderson's algorithm, and gave a quite complicated algorithm that uses "only" $O(n^{3/8}\text{polylog}n)$ colors. Recently, there has been a further step forward in this direction by Karger, Motwani, & Sudan (unpublished manuscript), who rely on an extension of the maximum cut algorithm of Goemans & Williamson (1994) to obtain a coloring with $O(n^{1/4}\log n)$ colors. They define a relaxation of the coloring problem called a *vector k-coloring*: each vertex v is assigned an n-dimensional unit vector $\vec{x}(v)$ such that for any two adjacent vertices u and v, $\vec{x}(u)\cdot\vec{x}(v) \leq -1/(k-1)$. The aim is find a vector k-coloring for as small a value of k as possible. It is not hard to show that any ordinary vertex coloring with k colors can be used to derive a vector k-coloring, by relying on the fact that there always exist k vectors whose pairwise inner products are all $-1/(k-1)$.

We give next the algorithm to round the vector k-coloring into a true coloring. This rounding relies on an intermediary relaxation of a coloring, where at most $n/3$ edges are permitted to have identical colors at their endpoints. Such a near-coloring must have a graph induced on $n/3$ nodes with a proper coloring, and so we can recursively find a good coloring on the remaining graph. Let Δ denote the maximum degree of the graph. The near-coloring is found by choosing $t = 1 + \log_3\Delta$ random hyperplanes, which divide the unit sphere into at most 2^t regions. The vector k-coloring is rounded to a near-coloring by assigning one color to the vectors in each region. In fact, this algorithm has a somewhat weaker performance guarantee than the one claimed above, but Karger, Motwani, & Sudan give a more sophisticated rounding technique to obtain a coloring with $O(n^{1/4}\log n)$.

Lund & Yannakakis (1993) showed that the problems of obtaining near-optimal cliques and near-optimal coloring were intimately connected. In essence, they showed that there is an approximation preserving reduction from the maximum clique problem (restricted to graphs of the sort just constructed by the proof above) to the minimum vertex coloring problem. Consequently, there exists an $\epsilon > 0$ such that no n^{ϵ}-approximation algorithm for vertex coloring exists, unless *P=NP*. As with the constant lower bounds for problems in *MAXSNP*, the specific values for ϵ implied by the proofs were initially extremely small and have steadily improved. Most recently, Bellare & Sudan (1994) showed that $\epsilon = .166$ is sufficient for the maximum clique problem, and that $\epsilon = .0714$ is sufficient for the minimum vertex coloring problem. The principle open question remaining is whether there exists some $\epsilon < 1$ such that an n^{ϵ}-approximation algorithm does exist for each of these two problems.

It remains a tantalizing open problem to give an algorithm that colors 3-colorable graphs with $O(\log n)$ colors. The best lower bounds are quite far from this level too. Khanna, Linial, & Safra (1993) show that it is not possible to give an algorithm that always uses 4 colors, unless *P=NP*.

Acknowledgments I would like to thank the people who have given me comments on preliminary versions of this paper: Leslie Hall, Dorit Hochbaum, David Johnson, Philip Klein, Jon Kleinberg, Paul Martin, Rajeev Motwani, Joel Wein, David Williamson, and Neal Young.

References

1. M. Aggarwal and N. Garg (1994). A scaling technique for better network design. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 233–239.
2. A. Agrawal, P. Klein, and R. Ravi (1991). When trees collide: an approximation algorithm for the generalized Steiner problem in networks. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 134–144. To appear in *SIAM J. Comput.*
3. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy (1992). Proof verification and hardness of approximation problems. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 14–23.
4. S. Arora and S. Safra (1992). Probabilistic checking of proofs. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–13.
5. L. Babai, L. Fortnow, and C. Lund (1991). Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity 1*, 3–40.
6. M. Balinski and K. Spielberg (1969). Methods for integer programming: algebraic, combinatorial, and enumerative. In J. Aronofsky, editor, *Progress in Operations Research, III*, pages 195–292. Wiley, New York.
7. R. Bar-Yehuda and S. Even (1981). A linear time approximation algorithm for the weighted vertex cover problem. *J. Algorithms 2*, 198–203.
8. M. Bellare, S. Goldwasser, C. Lund, and A. Russell (1993). Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 294–304.
9. M. Bellare and M. Sudan (1994). Improved non-approximability results. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 184–193.
10. P. Berman and V. Ramaiyer (1992). Improved approximations for the Steiner tree problem. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 325–334.
11. P. Berman and G. Schnitger (1992). On the complexity of approximating the independent set problem. *Information and Computation 96*, 77–94.
12. M. Bern and P. Plassman (1989). The Steiner problem with edge lengths 1 and 2. *Inform. Process. Lett. 32*, 171–176.

13. A. L. Blum (1991). *Algorithms for approximate graph coloring.* PhD thesis, MIT, Cambridge, MA.
14. R. B. Boppana and M. M. Halldórsson (1992). Approximating maximum independent sets by excluding subgraphs. *BIT 32*, 180–196.
15. N. Christofides (1976). Worst case analysis of a new heuristic for the travelling salesman problem. Technical Report 338, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.
16. F. R. K. Chung and S. T. Yau (1994). A near optimal algorithm for edge separators. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 1–8.
17. V. Chvátal (1979). A greedy heuristic for the set covering problem. *Math. Oper. Res. 4*, 233–235.
18. D.-Z. Du and F. K. Hwang (1990). An approach for proving lower bounds: solution of Gilbert-Pollak's conjecture on Steiner ratio. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 76–85.
19. D.-Z. Du, Y. Zhang, and Q. Feng (1991). On better heuristic for Euclidean Steiner minimum trees. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 431–439.
20. M. Dyer and A. Frieze (1985). A simple heuristic for the p-center problem. *Oper. Res. Lett. 3*, 285–288.
21. K. Eisemann (1957). The trim problem. *Management Science 3*, 279–284.
22. R. Fagin (1974). Generalized first-order spectra, and polynomial-time recognizable sets. In R. Karp, editor, *Complexity of Computations, AMS Symposia in Applied Mathematics 7*, AMS, 43–73.
23. T. Feder and D. H. Greene (1988). Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 434–444.
24. U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy (1991). Approximating clique is almost NP-complete. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 2–12.
25. U. Feige and L. Lovász (1992). Two-prover one-round proof systems: their power and their problems. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 733–744.
26. W. Fernandez de la Vega and G. S. Lueker (1981). Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica 1*, 349–355.
27. L. R. Ford, Jr. and D. R. Fulkerson (1956). Maximal flow through a network. *Canadian J. Math 8*, 399–404.
28. A. M. Frieze, G. Galbiati, and F. Maffioli (1982). On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks 12*, 23–39.
29. M. Fürer and B. Raghavachari (1992). Approximating the minimum degree spanning tree to within one from the optimal degree. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 317–324.
30. H. N. Gabow, M. X. Goemans, and D. P. Williamson (1993). An efficient approximation algorithm for the survivable network design problem. In *Proceedings of the 3rd MPS Conference on Integer Programming and Combinatorial Optimization*, pages 57–74.
31. M. R. Garey and D. S. Johnson (1976). The complexity of near-optimal graph coloring. *J. Assoc. Comput. Mach. 23*, 43–49.
32. N. Garg, V. V. Vazirani, and M. Yannakakis (1993). Approximate max-flow min-(multi)cut theorems and their applications. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 698–707.
33. E. N. Gilbert and H. O. Pollak (1968). Steiner minimal trees. *SIAM J. Appl. Math. 16*, 1–29.
34. M. X. Goemans and D. J. Bertsimas (1993). Survivable networks, linear programming relaxations, and the parsimonious property. *Math. Programming 60*, 145–166.

35. M. X. Goemans, A. V. Goldberg, S. A. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson (1994). Improved approximation algorithms for network design problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232.
36. M. X. Goemans and D. P. Williamson (1992). A general approximation technique for constrained forest problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 307–316. To appear in *SIAM J. Comput.*
37. M. X. Goemans and D. P. Williamson (1993). A new 3/4-approximation algorithm for MAX SAT. In *Proceedings of the 3rd MPS Conference on Integer Programming and Combinatorial Optimization*, 313–321. To appear in *SIAM J. on Discrete Math.*.
38. M. X. Goemans and D. P. Williamson (1994). .878-approximation algorithm for MAX CUT and MAX 2SAT. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 422–431.
39. R. L. Graham (1966). Bounds for certain multiprocessing anomalies. *Bell System Tech. J. 45*, 1563–1581.
40. M. M. Halldórsson (1993). A still better performance guarantee for approximate graph coloring. *Inform. Proc. Lett. 45*, 19–23.
41. D. S. Hochbaum (1982). Approximation algorithms for set covering and vertex cover problems. *SIAM J. Comput. 11*, 555–556.
42. D. S. Hochbaum and D. B. Shmoys (1985). A best possible approximation algorithm for the k-center problem. *Math. Oper. Res. 10*, 180–184.
43. I. Holyer (1981). The NP-completeness of edge-coloring. *SIAM J. Comput. 10*, 718–720.
44. W. L. Hsu and G. L. Nemhauser (1979). Easy and hard bottleneck location problems. *Discrete Appl. Math. 1*, 209–216.
45. F. K. Hwang (1976). On steiner minimal trees with rectilinear distance. *SIAM J. Appl. Math. 30*, 104–114.
46. D. S. Johnson (1974a). Fast algorithms for bin-packing. *J. Comput. System Sci. 8*, 272–314.
47. D. S. Johnson (1974b). Approximation algorithms for combinatorial problems. *J. Comput. System Sci. 9*, 256–278.
48. D. S. Johnson (1974c). Worst case behavior of graph coloring algorithms. In *Proceedings of the 5th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 513–527. Utilitas Mathematica Publishing, Winnipeg, Ont.
49. D. S. Johnson (1992). The NP-completeness column: an ongoing guide. *J. Algorithms 13*, 502–524.
50. N. Karmarkar and R. M. Karp (1982). An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 312–320.
51. S. Khanna, N. Linial, and S. Safra (1993). On the hardness of approximating the chromatic number. In *Proceedings of the 2nd Israeli Symposium on Theory and Computing Systems*, pages 250–260.
52. S. Khuller and U. Vishkin (1992). Biconnectivity approximations and graph carvings. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 759–770.
53. P. Klein, A. Agrawal, R. Ravi, and S. Rao (1990). Approximation through multicommodity flow. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 726–737. To appear in *Combinatorica*.
54. P. Klein and R. Ravi (1993). When cycles collapse: a general approximation technique for constrained two-connectivity problems. In *Proceedings of the 3rd MPS Conference on Integer Programming and Combinatorial Optimization*, pages 39–55.
55. T. Leighton and S. Rao (1988). An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 422–431.
56. J. K. Lenstra and A. H. G. Rinnooy Kan (1978). The complexity of scheduling under precedence constraints. *Operations Res. 26*, 22–35.

57. J. K. Lenstra, D. B. Shmoys, and É. Tardos (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming 46*, 259–271.
58. J. H. Lin and J. S. Vitter (1992). ϵ-approximations with minimum packing constraint violation. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 771–782.
59. L. C. Lorentzen (1966). Notes on covering of arcs by nodes in an undirected graph. Technical Report ORC 66.16, University of California, Berkeley.
60. L. Lovász (1975). On the ratio of optimal integral and fractional covers. *Discrete Math. 13*, 383–390, 1975.
61. C. Lund and M. Yannakakis (1993). On the hardness of approximating minimization problems. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 286–293.
62. G. L. Nemhauser and L. E. Trotter, Jr. (1975). Vertex packing: structural properties and algorithms. *Math. Programming 8*, 232–248.
63. C. H. Papadimitriou (1994). *Computational complexity.* Addison Wesley, Reading, MA.
64. C. H. Papadimitriou and M. Yannakakis (1991). Optimization, approximation, and complexity classes. *J. Computer Sys. Sciences 43*, 425–440.
65. C. H. Papadimitriou and M. Yannakakis (1993). The traveling salesman problem with distances one and two. *Math. Oper. Res. 18*, 1–11.
66. C. N. Potts (1985). Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Appl. Math. 10*, 155–164.
67. P. Raghavan and C. D. Thompson (1987). Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica 7*, 365–374.
68. D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput. 6*, 563–581.
69. S. Sahni and T. Gonzalez (1976). P-complete approximation problems. *J. Assoc. Comput. Mach. 23*, 555–565.
70. D. B. Shmoys and É. Tardos (1993). An improved approximation algorithm for the generalized assignment problem. *Mathematical Programming 62*, 461–474.
71. D. B. Shmoys and D. P. Williamson (1990). Analyzing the Held-Karp TSP bound: a monotonicity property with application. *Information Proc. Lett. 35*, 281–285.
72. É. Tardos (1993). Approximate min-max theorems and fast approximation algorithms for multicommodity flow problems. In *Summer School on Combinatorial Optimization*, pages 43–53.
73. M. A. Trick (1991). Scheduling multiple variable-speed machines. Unpublished manuscript.
74. V. G. Vizing (1964). On an estimate of the chromatic class of a p-graph (in Russian). *Diskret. Analiz 3*, 23–30.
75. A. Wigderson (1983). Improving the performance guarantee for approximate graph coloring. *J. Assoc. Comput. Mach. 30*, 729–735.
76. D. P. Williamson, M. X. Goemans, M. Mihail, and V. V. Vazirani (1993). A primal-dual approximation algorithm for generalized Steiner network problems. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 708–717. To appear in *Combinatorica.*
77. L. A. Wolsey (1980). Heuristic analysis, linear programming, and branch and bound. *Math. Programming Stud. 13*, 121–134.
78. M. Yannakakis (1992). On the approximation of maximum satisfiability. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9. To appear in *J. Algorithms.*
79. M. Yue (1991). A simple proof of the inequality $FFD(L) \leq 11/9OPT(L) + 1, \forall L$, for the FFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica 7*, 321–331.
80. A. Z. Zelikovsky (1993). An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica 9*, 463–470.

School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853

E-mail address: shmoys@cs.cornell.edu