Randomized Approximation Algorithms in Combinatorial Optimization

Prabhakar Raghavan *

August 9, 1994

Abstract

Randomization has proved to be a powerful technique in finding approximate solutions to difficult problems in combinatorial optimization. In this paper, we restrict ourselves to approximation algorithms that are efficient, and provably good. The focus of this paper is the use of *randomized rounding*. In this approach, one solves a relaxation of a problem in combinatorial optimization, and then uses randomization to return from the relaxation to the original optimization problem. Two kinds of relaxations of difficult combinatorial problems are considered: linear programming relaxations, and semidefinite programming relaxations. A number of concrete applications are given. This paper does not treat the very interesting applications of randomization in approximation algorithms for counting problems.

1. Introduction

The last few years have witnessed the proliferation of randomization in approximation algorithms for difficult combinatorial problems. Probabilistic search techniques such as simulated annealing [43] have enjoyed considerable success in solving large instances of a variety of combinatorial problems. In

*IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA.

this paper, we study approximation algorithms that are *efficient* in that their running times are (provably) bounded by a polynomial in the size of the input, and *provably good* in that the solution produced by the algorithm is guaranteed to be close to the optimal solution to within a specified, provable tolerance. Further, we insist that these guarantees hold for every instance of the problem being solved; the only randomness in the performance guarantee stems from the randomization in the algorithm itself, and not due to any probabilistic assumptions on the instance.

Among such provably good and efficient randomized algorithms, there have been two major areas of success. The first is that of approximation algorithms for combinatorial problems such as graph coloring, multicommodity flow and finding large cuts in a graph. This will be the focus of this paper. The second area, which we will not touch upon, is that of approximately counting the number of distinct solutions to an instance of a combinatorial problem — for example, the number of perfect matchings in a given graph [39]. These problems have applications in statistical physics and reliability theory.

All the algorithms we will discuss follow a common paradigm. They first formulate the problem as an integer program. Next, some constraints in this integer program are relaxed in order that the relaxation be efficiently solvable. Finally, randomization is used to restore the relaxed constraints.

In Section 2 we study linear program relaxations, using an integer multicommodity flow problem and the maximum satisfiability problem as illustrations. In Section 3, we study the application of semidefinite program relaxations to maximum cuts in a graph, and to graph coloring. We give a number of ways in which these results can be extended, and the results of some implementations, in Section 4. Throughout this paper, $\mathbf{E}[X]$ will denote the expectation of a random variable X, and $\mathbf{P}[A]$ will denote the probability of an event A. We will consider optimization problems that have objective functions whose value we seek to either minimize or maximize. We say that an algorithm is α-approximation algorithm if the solution it delivers has an objective function value that is at most α times the optimal solution, on every instance.

2. Rounding Linear Programs

In this section we discuss approximations obtained using the linear programming relaxation of the integer program formulation of an optimization problem. A convenient abstraction for beginning this study is the *lattice-approximation problem.* After studying this problem in the abstract, we will point out its connections to linear programming relaxations and to approximation algorithms in a number of concrete settings. In the lattice approximation problem, we are given an $n \times n$ matrix $\mathbf{A}$ all of whose entries are 0 or 1. In addition, we are given a column vector $\mathbf{p}$ with n entries, all of which are in the interval $[0, 1]$. We wish to find a column vector $\mathbf{q}$ with n entries, all of which are from the set $\{0, 1\}$, so as to minimize $\| \mathbf{A} \cdot (\mathbf{p} - \mathbf{q}) \|_\infty$. We think of the vector $\mathbf{q}$ as an "integer approximation" to the given real vector $\mathbf{p}$, in the sense that $\mathbf{A} \cdot \mathbf{q}$ is close to $\mathbf{A} \cdot \mathbf{p}$ in every component. Below, we apply the technique of *randomized rounding* to this problem. Following this, we give two instructive applications of the technique in detail, and finally mention a number of other interesting applications.

One solution to this problem is *thresholding:* for all i, if $p_i \geq 0.5$ set q_i to 1, else set it to 0. Thresholding may yield a vector $\mathbf{q}$ for which $\| \mathbf{A} \cdot (\mathbf{p} - \mathbf{q}) \|_\infty$ may be as large as $\| \mathbf{A} \cdot \mathbf{q} \|_\infty$. Consider the following randomized rounding scheme for determining the components of $\mathbf{q}$: for each i, independently set q_i to 1 with probability p_i, and to 0 otherwise. Then, letting $\mathbf{A}_i$ denote the ith row of $\mathbf{A}$, we have $\mathbf{E}[\mathbf{A}_i \cdot \mathbf{q}] = \mathbf{E}[\mathbf{A}_i \cdot \mathbf{p}]$ by linearity of expectation (this does not require the entries of $\mathbf{A}$ to be 0-1). We now argue that for all i, $\mathbf{A}_i \cdot \mathbf{q}$ is likely to remain close to $\mathbf{A}_i \cdot \mathbf{p}$ after randomized rounding. To this end we invoke the following bounds (commonly known as *Chernoff bounds* [29]) on the sum of independent 0-1 (or, *Bernoulli*) random variables.

Lemma 1: Let $X_1, \ldots, X_n$ be a sequence of independent Bernoulli trials such that $\mathbf{P}[X_i = 1] = p_i$ and $\mathbf{P}[X_i = 0] = 1 - p_i$. Let S be any subset of the integers $1, \ldots, n$, and let $s = |S|$. Define $Y = \sum_{i \in S} X_i$, so that $\mathbf{E}[Y] = \sum_{i \in S} p_i$. Then,

$$\mathbf{P}[|Y - \mathbf{E}[Y]| > \sqrt{4s \ln s}] \leq 1/s^2.$$

The following alternative version of the Chernoff bound [29] will also be useful in the sequel:

Lemma 2: Let $X_1, \ldots, X_n$ be a sequence of independent Bernoulli trials such that $\mathbf{P}[X_i = 1] = p_i$ and $\mathbf{P}[X_i = 0] = 1 - p_i$. Define $Y = \sum X_i$, so that $\mathbf{E}[Y] = \sum p_i$. Then, for $\beta \in [0, 1]$,

$$\mathbf{P}[|Y - \mathbf{E}[Y]| > \beta \mathbf{E}[Y]] \leq 2 \exp(-0.38\beta^2 \mathbf{E}[Y]).$$

By Lemma 1, we know that

$$\mathbf{P}[|\mathbf{A}_i \cdot \mathbf{q} - \mathbf{A}_i \cdot \mathbf{p}| > \sqrt{4n \ln n}] < 1/n^2.$$

Now,

$$\begin{aligned} & \mathbf{P}[\| \mathbf{A} \cdot (\mathbf{p} - \mathbf{q}) \|_\infty > \sqrt{4n \ln n}] \\ = \; & \mathbf{P}[\cup_{i=1}^n |\mathbf{A}_i \cdot \mathbf{q} - \mathbf{A}_i \cdot \mathbf{p}| > \sqrt{4n \ln n}] \\ \leq \; & \sum_{i=1}^n \mathbf{P}[|\mathbf{A}_i \cdot \mathbf{q} - \mathbf{A}_i \cdot \mathbf{p}| > \sqrt{4n \ln n}] \\ < \; & 1/n. \end{aligned}$$

We thus have:

Theorem 3: For every instance $\{\mathbf{A}, \mathbf{p}\}$ of the lattice approximation problem, randomized rounding finds a solution $\mathbf{q}$ such that $\| \mathbf{A} \cdot (\mathbf{p} - \mathbf{q}) \|_\infty \leq \sqrt{4n \ln n}$, with probability at least $1 - 1/n$.

Using slightly stronger versions of the Chernoff bound than Lemma 1 [35, 36, 38], it is possible to get slightly sharper results than that in Theorem 3. However, in the worst case, we know of no efficient algorithm that can find a solution $\mathbf{q}$ such that $\| \mathbf{A} \cdot (\mathbf{p} - \mathbf{q}) \|_\infty$ is $o(\sqrt{4n \ln n})$ on every instance.

The *set-balancing problem* [40, 41] is a special case of the lattice-approximation problem, in which every entry of $\mathbf{p}$ is 0.5. This problem has a rich history in combinatorics, and additionally has a direct application to a problem in integrated circuit design [11]. Gao and Kaufmann [14] show that a closely related problem arises in the solution of channel-routing problems in VLSI. Theorem 3 implies a solution to the set-balancing problem; directions for improving this solution will be suggested below in Section 4.4.

What does the lattice approximation problem have to do with linear programs and approximation algorithms? Given a combinatorial optimization problem, we first formulate it as a zero-one integer linear program, if possible. We do not know of good ways of solving large zero-one linear programs in general, so we first relax the integrality constraints, and solve the resulting linear program. The linear program can be solved by any one of several algorithms with proven theoretical and/or practical efficiency. Let $\mathbf{x}$ denote the vector of variables in the integer linear program formulation, and let $\widehat{\mathbf{x}}$ denote the vector of solutions resulting from the linear program. We now apply randomized rounding to restore integrality to the variables: for each i, independently set $\mathbf{x}_i$ to 1 with probability $\widehat{\mathbf{x}}_i$, and to 0 otherwise. Let $\bar{\mathbf{x}}_i$ denote the resulting 0-1 vector of solutions to the integer program. Let $\mathbf{a}$ denote a row of the coefficient matrix of the linear program formulation. By linearity of expectation, $\mathbf{E}[\mathbf{a} \cdot \bar{\mathbf{x}}] = \mathbf{a} \cdot \widehat{\mathbf{x}}$. This suggests that the expected value of the left-hand side of any constraint in the linear program will satisfy the bound prescribed by the right-hand side. Below, we illustrate the use of this technique in two settings — integer multicommodity flows, and maximum satisfiability. These applications have been chosen because they are especially illustrative; a number of others are listed in Section 2.3.

2.1. Integer Multicommodity Flows

We are given a directed network (V, E) with a set V of nodes and a set E of arcs. Let m denote $|E|$ and n denote $|V|$. Associated with each each arc $e \in E$ is a *capacity*, denoted $c(e)$. In an instance of the *multicommodity flow problem*, we are given a set of k triplets (s_i, t_i, d_i). In each triplet, s_i and t_i are nodes in the network, while d_i is a positive integer *demand*. A solution is a set of *flows*, denoted $f_i(e)$, where we think of $f_i(e)$ as the amount of commodity i that flows through arc e. The flows are subject to the following restrictions:

1. All the flows $f_i(e)$ should be integral.

2. The incoming and outgoing flows of each commodity at each node should obey conservation constraints.

3. Capacity constraints: for all e, $\sum_i f_i(e) \leq c(e)$.

4. Demand constraints: we must ship d_i units of commodity i from s_i to t_i. In our notation, we write $\sum_{e \in A_i} f_i(e) \geq d_i$, where A_i is the set of arcs leaving s_i.

As stated above, it is possible for the problem to be infeasible — the capacities available may not suffice to sustain the demands. Consider the following optimization version of the problem, for which we seek approximation algorithms. Given the instance, we seek to maximize the total flow of all commodities, which is clearly $\sum_i \sum_{e \in A_i} f_i(e)$. Thus, some commodities may not have their demands satisfied in the process of this maximization.

We now proceed to discuss the case $d_i = 1$ for all i; the same technique applies to the general case $d_i \geq 1$ with minor modifications. Clearly this maximization problem can be written as a 0-1 linear program. By relaxing the integrality constraint $f_i(e) \in \{0, 1\}$ for all i, e, we obtain the relaxation linear program in which $f_i(e) \in [0, 1]$ for all i, e. This linear program can be solved efficiently; in fact, one can use more efficient and direct combinatorial algorithms [23, 33]. We solve instead a slightly modified linear program, one in which we set the capacity of every arc e to $(1 - \epsilon)c(e)$ for a positive constant $\epsilon < (\sqrt{5} - 1)/2$. We solve this modified linear program to obtain the fractional solutions $\widehat{f}_i(e)$, together with the total flow $\widehat{F} = \sum_i \sum_{e \in A_i} f_i(e)$. Let F denote the value of the total flow in the optimal *integral* solution. Then, $\widehat{F} \geq F(1 - \epsilon)$, and the fractional total flow in any arc is at most $(1 - \epsilon)$ times its capacity. Randomized rounding is now invoked as follows: for each commodity i, we independently perform a random walk from s_i to t_i that is guided by the fractional solutions $\widehat{f}_i(e)$. The random walk for commodity i is as follows: we begin at s_i by flipping a coin with probability of heads equal to $\sum_{e \in A_i} f_i(e) \leq 1$. If the coin comes up heads, we proceed with the random walk as described below; if not, we assign zero flow to commodity i. We now describe the general step of the random walk: suppose that we are at a node v. Let $A(v)$ denote the set of arcs leaving v. The random walk then chooses to proceed along arc $a \in A(v)$ with probability $\widehat{f}_i(a)/(\sum_{e \in A(v)} \widehat{f}_i(e))$. The walk for commodity i terminates on reaching t_i, which it must in at most $n - 1$ steps because the set of arcs with non-zero flow may be assumed (without loss of generality) to form a directed acyclic graph. The following lemma is easy to prove by induction:

Lemma 4: The probability that the random walk for commodity i traverses any arc e is equal to $\widehat{f}_i(e)$.

We are now ready for the main result.

Theorem 5: Suppose that we have an instance of multicommodity flow in which every capacity $c(e)$ is at least $5.2 \ln 4m$. Let ϵ be a positive constant less than $(\sqrt{5}-1)/2$. Then, with probability $1 - 1/m$, the above algorithm yields a integer solution of total flow at least $F(1-\epsilon)^2$ with probability at least $1 - 1/m - 2\exp(-0.38\epsilon^2 F)$, where F is the total flow in the optimal integer solution.

Proof: The algorithm always produces an integer solution, so it can only fail to meet the guarantees of the theorem because of the following types of failure.

1. The rounded flow violates the capacity constraint on some arc.
2. The rounded flow is of value less than $F(1-\epsilon)^2$.

We proceed to bound the probability of the first mode of failure by $1/m$, and the second by $2\exp(-0.38\epsilon^2 F)$. Adding together these probabilities yields the theorem.

The crucial observation is that the event "the random walk for commodity i traverses arc e" is a Bernoulli trial, for any fixed arc e. Likewise, the event "commodity i has non-zero flow in the solution" is also a Bernoulli trial. Then, after randomized rounding the flow in any arc is the sum of independent Bernoulli trials. Likewise, the total flow is a random variable that is the sum of independent Bernoulli trials. We have already observed that the expectation of the total flow after rounding is at least $F(1-\epsilon)$. Applying Lemma 2 with $\beta = \epsilon$ now yields the probability bound for the second mode of failure. We turn now to the flow through a fixed arc of the network, following randomized rounding. Once again, Lemma 2 yields that the probability that the capacity constraint of a fixed arc is violated is at most $1/m^2$; we omit the detailed but routine calculations. Summing this probability over all m arcs, we have the desired bound. □ □

Note that the failure probability of the algorithm can be diminished by independent repetitions; in most applications, it is the computation of the fractional solutions $\widehat{f}_i(e)$ that is computationally intensive. Randomized rounding is itself quite fast, and can be repeated many times.

2.2. Maximum satisfiability

In the satisfiability problem: given a set of clauses in conjunctive normal form, we wish to decide whether there is an assignment of the variables that satisfies all the clauses. In the MAX-SAT problem, we are given an instance of satisfiability and we seek an assignment that maximizes the number of satisfied clauses. MAX-SAT is known to be NP-Hard. Given an instance I, let $M_*(I)$ be the maximum number of clauses that can be satisfied, and let $M_A(I)$ be the number of clauses satisfied by an algorithm A. The performance ratio of algorithm A is defined to be the infimum (over all instances I) of $M_A(I)/M_*(I)$. If A achieves a performance ratio of α, we call it an α-approximation algorithm. For a randomized algorithm A, the quantity $M_A(I)$ may be a random variable, in which case we replace $M_A(I)$ by $\mathbf{E}[M_A(I)]$ in the definition of the performance ratio. Thus our definition requires us to satisfy a number of clauses close to the best possible for the instance at hand. We now give a simple randomized algorithm that achieves a performance ratio of 3/4.

Consider setting each variable in the instance independently to 1 with probability 1/2. Clearly, a clause containing k literals is not satisfied by this process with probability $1/2^k$; thus such a random assignment would work well for instances in which every clause contains many literals: if every clause were to contain k or more literals, we immediately have a α-approximation algorithm for which $\alpha \geq 1 - 2^{-k}$. This idea is implicit in early work of Johnson [19]. It follows that we have a randomized 3/4-approximation algorithm for instances of MAX-SAT in which every clause has at least two literals. It appears that the bottleneck for achieving a performance ratio of 3/4 stems from clauses consisting of a single literal. We now give a different algorithm that performs especially well when there are many clauses consisting of single literals. This algorithm is due to Goemans and Williamson [15].[1] We then

[1]Prior to the work of Goemans and Williamson, Yannakakis [45] had given a determin-

argue that on any instance, one of these two algorithms will yield a (randomized) 3/4-approximation. Thus, given an instance, we run both algorithms and take the better of the two solutions. In Section 3 we will describe an improved algorithm due to Goemans and Williamson [16] that achieves an approximation ratio of 0.878.

The idea again is to formulate the problem as an integer linear program, solve the linear programming relaxation, and then to round using the randomized rounding. To each clause C_j in the instance, we associate an indicator variable $z_j \in \{0,1\}$ in the integer linear program that is 1 when C_j is satisfied and 0 otherwise. For each variable x_i, we use an indicator variable y_i that is 1 if the variable x_i is set TRUE, and 0 otherwise. Let S_j^+ be the set of variables that appear in the uncomplemented form in clause S_j, and S_j^- be the set of variables that appear in the complemented form in clause C_j. We may then formulate the MAX-SAT problem as follows:

$$\text{Maximize } \sum_j z_j \text{ subject to}$$

$$\sum_{S_j^+} y_i + \sum_{S_j^-} (1 - y_i) \geq z_j \qquad \forall j,$$

$$y_i, z_j \in \{0,1\}, \qquad \forall i, j. \tag{1}$$

We solve the relaxation linear program in which we relax the integrality constraints (1), allowing the y_i and the z_j to assume values in the interval $[0,1]$. Let $\widehat{y}_i$ be the value assigned to y_i in the optimal solution to this linear program, and let $\widehat{z}_j$ be the value assigned to z_j. Clearly $\sum_j \widehat{z}_j$ is an upper bound on the number of clauses that can be satisfied. We first show that using randomized rounding, we obtain a solution in which the expected number of clauses satisfied $\geq (1 - 1/e) \sum_j \widehat{z}_j$.

Randomized rounding independently sets the variable y_i to 1 (corresponding to x_i being set TRUE) with probability $\widehat{y}_i$. Where k is a positive integer, let β_k denote $1 - (1 - 1/k)^k$. We first show that for a clause C_j with k literals, the probability that it is satisfied by randomized rounding is at least

istic polynomial-time 3/4-approximation algorithm for MAX-SAT.

$\beta_k \widehat{z}_j$. Because $\beta_k \geq (1-1/e)$, the expected number of clauses satisfied by randomized rounding is at least $(1-1/e)\sum_j \widehat{z}_j$.

Lemma 6: Let C_j be a clause with k literals. The probability that it is satisfied by randomized rounding is at least $\beta_k \widehat{z}_j$.

Proof: Consider a single clause C_j. We may assume without loss of generality that all the variables contained in it appear in uncomplemented form, and that C_j is of the form $x_1 \vee \cdots \vee x_k$. By constraint (1) in the linear program,

$$\widehat{y}_1 +, \cdots, +\widehat{y}_k \geq \widehat{z}_j.$$

Clause C_j remains unsatisfied by randomized rounding only if every one of the variables y_i is rounded to 0. Since each variable is rounded independently, this occurs with probability $\prod_{i=1}^{k}(1-\widehat{y}_i)$. To show that

$$1 - \prod_{i=1}^{k}(1-\widehat{y}_i) \geq \beta_k \widehat{z}_j,$$

we observe that the expression on the left is minimized when $\widehat{y}_i = \widehat{z}_j/k$ for all i. It now suffices to show that $1-(1-z_j/k)^k \geq \beta_k z_j$ in $[0,1]$; this follows from elementary calculus. □ □

Theorem 7: Given an instance of MAX-SAT, the expected number of clauses satisfied by linear programming and randomized rounding is at least $(1-1/e)$ times the maximum number of clauses that can be satisfied on that instance.

We have studied two randomized algorithms MAX-SAT: one that set each variable to 1 with probability 1/2, and a second that used the solutions to the linear program as a basis for randomized rounding. A simple convexity argument shows that in fact on any instance, one of the algorithms is a 3/4-approximation algorithm; the reader may reconstruct the calculation from the form of the statement of the following theorem. Given any instance, we run both algorithms and choose the better solution. Let n_1 denote the expected number of clauses that are satisfied when each variable is independently set to 1 with probability 1/2. Let n_2 denote the expected number of clauses that are satisfied when we use the linear programming followed by randomized rounding (corresponding to Theorem 7).

Theorem 8:

$$\max\{n_1, n_2\} \geq \frac{3}{4} \sum_j \widehat{z}_j.$$

2.3. Related Work

Space precludes our including many other interesting applications of randomized rounding. Naor and Roth [30] apply the technique to a file distribution problem in networks. Klein and Sairam [22] apply it to the computation of approximate shortest paths in a graph. Lin and Vitter [25] consider a family of packing problems. Kortsarz and Peleg [24] study the construction of a useful class of graphs known as *spanners*. The paper by Bertsimas and Vohra [6] is a comprehensive guide to uses of the technique in a variety of covering problems. One of the interesting ideas in their paper is to devise a function that maps each value in the linear program solution to a probability in $[0, 1]$; in the above examples, we have only used the obvious identity function. In some applications, as shown in [6], it helps to use other mappings from fractional solutions to probabilities. The interested reader is referred to these papers and the ones cited in them, for further applications. Before proceeding to semidefinite programming, it is worth pointing out an alternative view of randomized rounding. Instead of thinking of the process as rounding a fraction $\hat{x}$ to 1 with probability $\hat{x}$, we may adopt the following equivalent view. We pick a random number y uniformly in $[0, 1]$. If $\hat{x} > y$ we round up to 1, otherwise we round down to 0. The effect is the same, but we now think of comparing the fraction $\hat{x}$ to a randomly chosen threshold. This view will be useful in the following section.

3. Semidefinite Programming

The linear programming relaxations of Section 2 have the property that the value of the optimum for the relaxation is never far from the value of the optimum for the original integer program. Given this, we were able to obtain good approximations to the original integer program by finding an integer solution whose value is close to the optimum for the linear program. For many problems in combinatorial optimization, the linear program and integer program optima can be quite different. For instance, in the multicommodity flow problem of Section 2.1, if all arc capacities are 0 or 1, it is easy to construct instances for which the optimal value for the linear program is $\sqrt{n}$ times the optimal value for the integer program (in fact, the optimal flow values in the linear program solution need only assume values in the set $\{0, 0.5, 1\}$, so that the solutions are quite "discrete"). One approach to coping with this large discrepancy between the optima of the original problem and the relaxation is to consider relaxations other than linear programming relaxations. In this direction, the use of *semidefinite program* relaxations has recently enjoyed striking success. We begin by reviewing some of the main ideas behind semidefinite programming. We then illustrate the use of such relaxations using two examples.

We say that an $n \times n$ matrix $\mathbf{A}$ of reals is *positive semidefinite* if, for any vector $\mathbf{x} \in \Re^n$, $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$. It is known [17] that for any symmetric positive semidefinite matrix $\mathbf{A}$, there exists a matrix $\mathbf{B}$ such that $\mathbf{B}^T\mathbf{B} = \mathbf{A}$. It is further known [17] that $\mathbf{B}$ is an $m \times n$ matrix for $m \leq n$, and given $\mathbf{A}$ such a matrix $\mathbf{B}$ of full row-rank can be found in time polynomial in n. When all the diagonal entries of $\mathbf{A}$ are equal to 1, each row of $\mathbf{B}$ is a unit vector in $\Re^m$.

In an instance of a semidefinite programming problem, we wish to optimize a linear function of (the entries of) a symmetric matrix $\mathbf{A}$ subject to linear constraints and the additional constraint that $\mathbf{A}$ be positive semidefinite. There has been considerable work on the theory of semidefinite programming; the fact we require here is that for any positive real ϵ, a semidefinite program can be solved to within an additive error of ϵ in time polynomial in the size of the input and $\log 1/\epsilon$ [1, 18, 27]. We now proceed to our first application of semidefinite programming.

3.1. The MAX-CUT problem

In the MAX-CUT problem, we are given an undirected graph $G = (V, E)$ with vertex set V and edge set E. We wish to partition the vertices into two sets V_- and V_+ so as to maximize the number of edges with one end-point in V_- and one end-point in V_+. We refer to such a partition as a *cut*, and the edges between vertices in V_- and vertices in V_+ as *cut-edges*. The problem has applications in statistical physics and in VLSI design, and is NP-hard. Sahni and Gonzalez [37] give a 1/2-approximation algorithm for this problem: an algorithm that, on every instance, finds a cut such that number of cut-edges is at least half the number in the optimal cut. Indeed, this performance is achieved by the following *greedy algorithm*: initially, partition the vertices arbitrarily; thereafter, consider each vertex in turn, moving it to the other partition if this increases the number of edges in the cut. We now present an elegant application of semidefinite programming followed by randomized rounding to this problem, which yields a 0.878-approximation algorithm. This algorithm is due to Goemans and Williamson [16].

Consider first the following quadratic integer programming formulation of MAX-CUT. We use a variable x_i to indicate which side of the partition the ith vertex of V goes into. Let a_{ij} be 1 if the edge (i,j) exists, and 0 otherwise.

$$\text{Maximize } \sum_{i<j} a_{ij}(1 - x_i x_j)/2,$$
$$\text{subject to } x_i \in \{-1, +1\}\ \forall i. \tag{2}$$

Clearly the objective function measures the number of edges in the resulting cut. We now relax this reformulation using a semidefinite program. Viewing the allowable values of x_i in (2) as unit vectors in one dimension, the relaxation consists of allowing the x_i to be unit vectors in dimension $m \leq n$. We interpret the product $x_i x_j$ of the objective function in (2) as a dot product when considering vectors in higher dimensions. The resulting relaxation becomes:

$$\text{Maximize } \sum_{i<j} a_{ij}(1 - \mathbf{v}_i \cdot \mathbf{v}_j)/2,$$
$$\text{subject to } \mathbf{v}_i \in S_n\ \forall i, \tag{3}$$

where S_n denotes the set of unit vectors in n dimensions. Letting y_{ij} denote $\mathbf{v}_i \cdot \mathbf{v}_j$, we can rewrite (3) as

$$\text{Maximize } \sum_{i<j} a_{ij}(1 - y_{ij})/2, \text{ subject to}$$
$$y_{ii} = 1,\ \forall i \text{ and } \mathbf{Y} \text{ must be symmetric positive semidefinite.} \tag{4}$$

The correspondence between the formulations (3) and (4) is clear: a solution to (3) immediately yields a solution to (4); further, given a solution to (4), an incomplete Cholesky decomposition can be used to reconstruct the vectors $\mathbf{v}_i$ [17], without altering the objective function value. Noting that (4) is a semidefinite program, we can solve it in polynomial time (to within any desired precision) using the methods of [1, 18, 27]. The solution $\mathbf{Y}$ can now be factored into the sets of vectors $\mathbf{v}_i$, and it remains to round these vectors to $\{-1, +1\}$. To this end we use the following randomized rounding procedure: we pick a random unit vector $\mathbf{r}$ in $\Re^n$. For each i, we declare x_i (formulation (2)) to be $\text{sgn}(\mathbf{v_i} \cdot \mathbf{r})$, where $\text{sgn}(z)$ is the sign of the real z, and is $+1$ if z is non-negative and -1 otherwise. An equivalent view of this process is that we pick a random hyperplane (whose normal is $\mathbf{r}$) through the origin in $\Re^n$, and use it as a "threshold": vertices corresponding to the vectors on the same side of it are deemed to be in the same partition of the cut. We turn to the quality of this solution.

Theorem 9: Let C denote the size of the cut found by the above algorithm. Then,

$$\mathbf{E}[C] \geq 0.878 \sum_{i<j} a_{ij}(1 - \mathbf{v}_i \cdot \mathbf{v}_j)/2. \tag{5}$$

Remark: Since the right hand side of (5) is the value of the optimum for the relaxation (3) (which is in turn an upper bound on the size of the optimal cut), we find a cut whose expected size is at least 0.878 times the optimal cut.

Proof: We first claim that for any two vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ and for a random unit vector $\mathbf{r}$, the probability that $\text{sgn}(\mathbf{v_i} \cdot \mathbf{r}) \neq \text{sgn}(\mathbf{v_j} \cdot \mathbf{r}) = \cos^{-1}(\mathbf{v_i} \cdot \mathbf{v}_j)/\pi$. This follows by noting that the set of vectors $\mathbf{r}$ (equivalently, the set of hyperplanes) that separate $\mathbf{v}_i$ and $\mathbf{v}_j$ lie on two spherical sectors on the unit sphere in $\Re^n$, each of measure $\cos^{-1}(\mathbf{v_i} \cdot \mathbf{v}_j)/2\pi$.

Now, by linearity of expectation and the fact that the $\mathbf{v}_i$ are solutions to (3), we have

$$\mathbf{E}[C] = \sum_{i<j} a_{ij} \mathbf{P}[\text{sgn}(\mathbf{v_i} \cdot \mathbf{r}) \neq \text{sgn}(\mathbf{v_j} \cdot \mathbf{r})], \tag{6}$$

which is clearly $\sum_{i<j}(a_{ij} \cos^{-1}(\mathbf{v_i} \cdot \mathbf{v}_j))/\pi$. The proof is completed by invoking elementary calculus to argue that for any $z \in \{-1, +1\}$, $\cos^{-1} z \geq 0.878(1 - z)$. Substituting this inequality in (6) yields the result. □ □

It follows from work of Delorme and Poljak [10] that this semidefinite programming relaxation cannot yield a substantially better result; the reader is referred to [16] for details. An approach very similar to the above leads to a 0.878-approximation algorithm for the MAX-2SAT problem: this is the same as the MAX-SAT problem, with the further restriction that every clause contain exactly two literals. This result is also described in [16], as is the resulting 0.755-approximation for MAX-SAT. Recently, Feige and Goemans [12] have extended these techniques to obtain a 0.931-approximation algorithm for MAX-2SAT.

3.2. Graph Coloring

Given an undirected graph, a *k-coloring* of G is an assignment of an integer in $[1, k]$ to each vertex of G; we call this integer the *color* of that vertex. Such a k-coloring is said to be *legal* if no two adjacent vertices are assigned the same color. The graph coloring problem seeks a legal k-coloring for as small a value of k as possible; the smallest such k is called the *chromatic number* of G. A graph G is said to be k-colorable if there exists a legal k-coloring of G. It is *NP*-hard to decide whether a graph is 3-colorable. The graph coloring problem has arises in time-tabling, register allocation in compilers and in resource allocation.

The subject of this section is a randomized approximation algorithm for graph coloring due to Karger, Motwani and Sudan [20]. Although their algorithm is quite general, for clarity and brevity we only describe their algorithm for 3-colorable graphs. For such graphs, Wigderson [44] gave the following algorithm that colors any 3-colorable graph using $O(\sqrt{n})$ colors. Any graph of maximum degree Δ can be colored using $\Delta + 1$ colors, using standard techniques from graph theory [8]. Using this observation as a starting point, Wigderson's algorithm colors any graph for which $\Delta \leq \sqrt{n}$ using $\sqrt{n} + 1$ colors. Thus, it remains to deal with graphs in which there is a vertex of degree exceeding $\sqrt{n}$. The key observation is that in a 3-colorable graph, the subgraph induced by the neighbors of any vertex is bipartite, and can be colored in polynomial time using 2 colors. Thus, a vertex of degree exceeding $\sqrt{n}$, together with its neighbors, can be colored using 3 colors. We repeatedly color and remove such vertices (and their neighbors) from the graph, expending 3 colors at each iteration. We either finish with the empty graph in at most $\sqrt{n}$ iterations (using $O(\sqrt{n})$ colors in the process), or with a graph of maximum degree $\sqrt{n}$ (this residual graph can now be colored with at most $\sqrt{n}+1$ colors as before). In either case, the number of colors used is $O(\sqrt{n})$.

Blum [7] improved on Wigderson's algorithm by devising an algorithm that uses $n^{3/8}\log^{O(1)} n$ colors. We now describe the approximation algorithm of [20] based semidefinite programming; it uses

$$O(\min\{\Delta^{1/3}\log^{4/3}\Delta, n^{1/4}\log n\})$$

colors, where Δ is the maximum degree of any vertex in the graph.

The relaxation proposed by [20] is the following: consider representing the color of each vertex by a unit vector in n dimensions. The idea is to constrain the vectors corresponding to colors of adjacent vertices to be "far apart". To this end, they define a *vector k-coloring* of G to be an assignment of unit vectors $\mathbf{u}_i$ in $\Re^n$ to the vertices of G so that for adjacent vertices i, j the corresponding vectors satisfy $\mathbf{u}_i \cdot \mathbf{u}_j \leq -1/(k-1)$. Let the matrix $\mathbf{M} = \{m_{ij}\}$ be defined by $m_{ij} = \mathbf{u}_i \cdot \mathbf{u}_j$. Given a positive semidefinite matrix $\mathbf{M}$, we call it a *matrix k-coloring* of G if we can extract from it a matrix $\mathbf{U}$ such that $\mathbf{U}\mathbf{U}^T = \mathbf{M}$, where the rows of $\mathbf{U}$ form a vector k-coloring. This definition is motivated by the following two assertions:

1. If G is k-colorable, it has a vector k-coloring. This follows from the fact that for any $k < n$, there exist k unit vectors in $\Re^n$ the dot product of any two of which is $-1/(k-1)$, and we may associate one of these unit vectors with each of the color classes in a k-coloring of G.

2. If G is k-colorable, a matrix k-coloring (and hence a vector k-coloring) for it can be found using semidefinite programming in time polynomial in n and $1/\epsilon$, to within an additive error ϵ. The semidefinite program formulation in fact has interesting connections to the work of Lovász on the relationship between the clique and chromatic numbers of a graph [18, 26, 27].

Thus, given a 3-colorable graph, we can efficiently compute a vector 3-coloring for it. We turn to rounding this solution. One approach to rounding it to a C-coloring for a positive integer C is to partition $\Re^n$ into C "octants" about the origin using $\log_2 C$ random hyperplanes through the origin. Vertices whose vectors fall into the same octant are deemed to be in the same color class. A problem with this approach is that after the rounding, there is a good chance that two adjacent vertices are assigned the same color class. To circumvent this problem Karger, Motwani and Sudan [20] propose the notion of a *semicoloring:* a k-semicoloring of $G = (V, E)$ is a coloring for which at most $|V|/3$ edges of G have both end-points in the same color class. Thus, given a k-semicoloring of a graph, a constant fraction of its vertices are assigned legal colors and can be put aside. Repeating this process yields the following lemma:

Lemma 10: Let $f(i)$ be a non-decreasing function on the positive integers. Given an algorithm that yields an $f(m)$-semicoloring on every subgraph of G with m vertices, it can be used to color G using $O(f(n) \log n)$ colors.

In fact, the number of colors in Lemma 10 can be bounded by $O(f(n))$ provided $f(n) = \Omega(n^\epsilon)$ for a constant $\epsilon > 0$.

Consider now a 3-colorable graph for which we have found a vector 3-coloring. We now show that partitioning $\Re^n$ into $O(\Delta^{\log_3 2})$ octants using random hyperplanes through the origin is likely to yield a $O(\Delta^{\log_3 2})$-semicoloring, and from this argue that we can obtain a $O(\Delta^{\log_3 2} \log n)$-coloring of G (recall that Δ is the maximum vertex degree in G). The vector 3-coloring ensures that the dot product of the vectors assigned to any two adjacent vertices is at most $-1/2$, so that the angle between them is at least $2\pi/3$ radians. As in the proof of Theorem 9, the probability that a single random hyperplane separates two adjacent vertices is thus at least $2/3$. The probability that an edge of G is not separated by any of $\lceil 1 + \log_3 \Delta \rceil$ random hyperplanes is thus at most $(1/3)^{\lceil 1+\log_3 \Delta \rceil} \leq 1/(3\Delta)$. Since the number of edges in G is at most $n\Delta/2$, the expected number of edges that are not cut is at most $n/6$. By the Markov inequality the probability that we have more than $n/3$ bad edges is less than $1/2$, so that the expected number of independent roundings before we have a valid $O(\Delta^{\log_3 2})$-semicoloring is at most 2.

Wigderson's algorithm [44] now yields a coloring of G using $O(n^{0.387})$ colors. Let $\delta = n^{0.613}$; if there is a vertex of degree greater than δ, color it and its neighbors using 3 colors (since G is 3-colorable, the neighborhood of any vertex is bipartite and thus 2-colorable in polynomial time). Repeating this process on the subgraph induced by the remaining vertices, we use $O(n/\delta)$ colors until we are left with either the empty graph or a graph with maximum degree less than δ. Once the maximum degree is down to δ, we can color the graph with $O(\delta^{\log_3 2})$ colors by rounding the above semidefinite program, for a total of $O(n/\delta + \delta^{\log_3 2}) = O(n^{0.387})$ colors.

This guarantee is slightly weaker than that of Blum [7], whose algorithm finds a $n^{3/8} \log^{O(1)} n$-coloring of a 3-colorable graph. Karger, Motwani and Sudan [20] extend the above ideas using a slightly different rounding scheme from the random hyperplane partition defined above, to obtain the following theorem:

Theorem 11: Given a k-colorable graph with maximum degree Δ, it is possible to find a semicoloring using $\Delta^{1-2/k} \log^{O(1)} n$ colors with high probability, in polynomial time.

The proof of Theorem 11 is accomplished using the following alternative rounding method. Pick C random vectors $\mathbf{v}_1, \ldots, \mathbf{v}_C$ in $\Re^n$ using the n-dimensional normal distribution (thus, these are not unit vectors). We assign vertex i (whose vector color is $\mathbf{u}_i$) to color class j if $\mathbf{u}_i \cdot \mathbf{v}_j \leq \mathbf{u}_i \cdot \mathbf{v}_\ell$ for all $\ell \neq j$ (ties are broken arbitrarily). The key step of the analysis (which is too lengthy to reproduce here) is to show that with this form of rounding, the probability that two adjacent vertices are assigned to the same color class is asymptotically $C^{-k/(k-2)}$, so that it is possible to use a value of C that is roughly $\Delta^{1-2/k}$. The reader is referred to [20] for details.

Frieze and Jerrum [13] have applied a similar technique to the MAX k-CUT and MAX-BISECTION problems in graphs.

4. Conclusion

In this section we discuss a number of issues related to the techniques and problems discussed above.

4.1. Hardness of Approximation

A series of recent breakthroughs has formalized the study of fundamental limits on the extent to which it is possible to approximate the solutions to combinatorial problems, using efficient algorithms [2, 3, 32]. Bellare and Sudan [4] give a number of such hardness results pertaining to the problems we have discussed (see also [21]). For instance, they show that unless $NP = RP$, it is impossible to approximate MAX-SAT to a factor better than 73/74 using any randomized polynomial-time algorithm. They also give a number of results on graph coloring; for instance, they show that unless $NP = \text{co-}RP$, it is impossible to approximate the chromatic number to a factor better than $n^{1/13-\epsilon}$ for any fixed $\epsilon > 0$, using a randomized polynomial-time algorithm.

4.2. Derandomization and Parallelization

Most of the algorithms described in this paper can be *derandomized* using standard techniques [35, 41] to yield deterministic polynomial-time algorithms that (on every instance and on every execution) yield approximations as good as those given by the randomized algorithms they are derived from. The sole exception is the coloring algorithm leading to Theorem 11. However, the process of derandomization typically takes a relatively simple and clean randomized rounding procedure and turns it into a complex and generally slower deterministic algorithm. In practice, thus, the randomized algorithm is likely to be used. At a slight loss in the approximation ratios achieved, these deterministic algorithms can be turned into parallel algorithms that use a polynomial number of processors and a polylogarithmic number of steps [5, 28]. The study of such parallel algorithms has led to a rich theory of pseudorandom numbers. It is unclear whether these derandomized sequential or parallel algorithms will have any impact on computational practice.

4.3. Computational Experience

The multicommodity flow algorithm of Section 2.1 has found application in global wiring in gate-arrays [36]. Experiments have shown that for relatively small gate-arrays, the method performs well [31]. For large, dense gate-arrays, the cost of solving the linear program becomes prohibitive [34], although the rounding remains extremely fast and fairly effective. For this problem as well as the set-balancing problem, the bounds given by Theorem 3 are pessimistic in comparison to what is empirically observed [11]. Goemans and Williamson [16] report that on random graphs, the MAX-CUT algorithm of Section 3.1 found cuts that were typically within 4% of the bound given by the semidefinite relaxation. In these experiments, they used the best of 50 roundings of the solution given by the semidefinite program solution.

4.4. Open Problems

We conclude with a set of open problems related to the techniques we have discussed.

Set-balancing: Devise a (deterministic or randomized) polynomial-time that will solve the set-balancing problem yielding a solution with discrepancy $o(\sqrt{n \log n})$ on every instance. Spencer [40] shows that there is always a solution with discrepancy at most $6\sqrt{n}$, but his proof does not yield an efficient algorithm.

Edge-disjoint paths: We are given an undirected graph G, together with a set of source-sink pairs (s_i, t_i). We wish to find a path joining each source s_i to its sink t_i, such that the resulting paths are edge-disjoint. In the maximization version of the problem, we seek to maximize the number of source-sink pairs that can be thus connected. It is clear that this is a special case of integer multicommodity flow, but the techniques of Section 2 do not seem to apply here. Indeed, it is easy to construct instances on n-vertex graphs in which the ratio of the optima of the LP relaxation and the integer program is $\Omega(\sqrt{n})$.

Independent sets and vertex covers: Given an undirected graph G, a subset of its vertices is said to be *independent* if no two of them are adjacent in G. The independent set problem is to find the largest independent set in G. A *vertex cover* is a subset S of the vertices of G such that every edge is incident to at least one vertex of S. The vertex cover problem seeks a vertex cover that is as small as possible. Both the independent set problem and the vertex cover problem are NP-hard. Given a vertex cover S for a graph G, the vertices of G *not* in S form an independent set (and vice versa). The best approximation algorithm known for the independent set problem achieves an approximation ratio of $O(n/(\log^2 n))$ [9]. Szegedy [42] has recently applied semidefinite programming to this problem. Bellare and Sudan [4] give evidence that it is unlikely that the independent set problem can be approximated to within a factor better than $n^{1/4}$. For the vertex cover problem, there is a simple approximation algorithm that achieves a ration of 2; no significant improvement on this is known.

Acknowledgements

The author thanks Rajeev Motwani, Seffi Naor and Madhu Sudan for their help in preparing this manuscript.

References

[1] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. To appear in the *SIAM Journal on Optimization*, 1994.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[3] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1992.

[4] M. Bellare and M. Sudan. Improved non-approximability results. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 184–193, 1994.

[5] B. Berger and J. Rompel. Simulating $(\log^c n)$-wise independence in NC. *Journal of the ACM*, 38:1026–1046, 1991.

[6] D. Bertsimas and R. Vohra. Linear programming relaxations, approximation algorithms and randomization: a unified view of covering problems. Technical Report OR 285-94, MIT, 1994.

[7] A. Blum. New approximation algorithms for graph coloring. *Journal of the ACM*, 41:470–516, 1994.

[8] J. A. Bondy and U.S. R. Murty. *Graph Theory With Applications*. American Elsevier, 1977.

[9] R.B. Boppana and M.M. Halldorson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32:180–196, 1992.

[10] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Mathematical Programming*, 62:557–574, 1993.

[11] G.S. Ditlow and P. Raghavan. Timing-driven partitioning of PLAs. *IBM Technical Disclosure Bulletin*, 31, 1989.

[12] U. Feige and M.X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. Submitted for publication, 1994.

[13] A. Frieze and M. Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. Submitted for publication, 1994.

[14] S. Gao and M. Kaufmann. Channel routing of multiterminal nets. In *28th Annual Symposium on Foundations of Computer Science*, pages 316–325, October 1987.

[15] M. X. Goemans and D. P. Williamson. New 3/4-approximation algorithms for MAX SAT. To appear in *SIAM J. Disc. Math.*, 1993.

[16] M.X. Goemans and D.P. Williamson. 0.878-approximation algorithms for MAX-CUT and MAX-2SAT. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 422–431, 1994.

[17] G.H. Golub and C.F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, 1983.

[18] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[19] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[20] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[21] S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. In *Proceedings of the 2nd Israeli Symposium on Theory and Computing Systems*, pages 250–260, 1993.

[22] P.N. Klein and S. Sairam. A parallel randomized approximation scheme for shortest paths. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 750–758, 1992.

[23] P.N. Klein, C. Stein, and É. Tardos. Leighton-Rao might be practical: faster approximation algorithms for concurrent flow with uniform capacities. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 310–321, 1990.

[24] G. Kortsarz and D. Peleg. Generating low-degree 2-spanners. Technical Report CS93-07, The Weizmann Institute of Science, 1993.

[25] J.-H. Lin and J.S. Vitter. ϵ-approximations with minimum packing constraint violation. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 771–782, 1992.

[26] L. Lovász. On the Shannon capacity of a graph. *IEEE Trans. Information Theory*, IT-25:1–7, 1979.

[27] L. Lovász M. Grötschel and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, 1987.

[28] R. Motwani, J. Naor, and M. Naor. The probabilistic method yields deterministic parallel algorithms. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 8–13, October 1989.

[29] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, 1994. In Press.

[30] M. Naor and R.M. Roth. Optimal file sharing in distributed networks. In *32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 515–525, 1991.

[31] A.P-C. Ng, P. Raghavan, and C.D. Thompson. Experimental results for a linear program global router. *Computers and Artificial Intelligence*, 6(3):229–242, 1987.

[32] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[33] S.A. Plotkin, D.B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 495–504, 1991.

[34] W.R. Pulleyblank and P. Raghavan, 1994. Personal communication, IBM Yorktown Heights.

[35] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, October 1988.

[36] P. Raghavan and C.D. Thompson. Randomized rounding. *Combinatorica*, 7:365–374, 1987.

[37] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976.

[38] J.P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 331–340, 1993.

[39] A. Sinclair. *Algorithms for Random Generation and Counting: A Markov Chain Approach.* Progress in Theoretical Computer Science. Birkhauser, Boston, 1992.

[40] J. Spencer. Six standard deviations suffice. *Transactions of the American Mathematical Society*, 289(2):679–706, June 1985.

[41] J. Spencer. *Ten Lectures on the Probabilistic Method*. SIAM, 1987.

[42] M. Szegedy. A note on the θ number of Lovász and the generalized Delsarte bound. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[43] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Mathematics and its Applications. Reidel Publishing Company, 1987.

[44] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30:729–735, 1983.

[45] M. Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9, 1992.