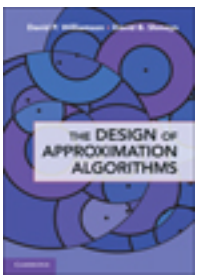

The Design of Approximation Algorithms

David P. Williamson, David B. Shmoys

Book DOI: http://dx.doi.org/10.1017/CBO9780511921735

Online ISBN: 9780511921735

Hardback ISBN: 9780521195270

Chapter

5 - Random Sampling and Randomized Rounding of Linear Programs pp. 99-136

Chapter DOI: http://dx.doi.org/10.1017/CBO9780511921735.006

Cambridge University Press

CHAPTER 5

Random Sampling and Randomized Rounding of Linear Programs

Sometimes it turns out to be useful to allow our algorithms to make random choices; that is, the algorithm can flip a coin, or flip a biased coin, or draw a value uniformly from a given interval. The performance guarantee of an approximation algorithm that makes random choices is then the *expected* value of the solution produced relative to the value of an optimal solution, where the expectation is taken over the random choices of the algorithm.

At first this might seem like a weaker class of algorithm. In what sense is there a performance guarantee if it holds only in expectation? However, in most cases we will be able to show that randomized approximation algorithms can be *derandomized*: that is, we can use a certain algorithmic technique known as the method of conditional expectations to produce a deterministic version of the algorithm that has the same performance guarantee as the randomized version. Of what use then is randomization? It turns out that it is often much simpler to state and analyze the randomized version of the algorithm than to state and analyze the deterministic version that results from derandomization. Thus, randomization gains us simplicity in our algorithm design and analysis, while derandomization ensures that the performance guarantee can be obtained deterministically.

In a few cases, it is easy to state the deterministic, derandomized version of an algorithm, but we know how to analyze only the randomized version. Here the randomized algorithm allows us to analyze an algorithm that we are unable to analyze otherwise. We will see an example of this when we revisit the prize-collecting Steiner tree problem in Section 5.7.

It is also sometimes the case that we can prove that the performance guarantee of a randomized approximation algorithm holds with high probability. By this we mean that the probability that the performance guarantee does not hold is one over some polynomial in the input size of the problem. Usually we can make this polynomial as large as we want (and thus the probability as small as we want) by weakening the performance guarantee by some constant factor. Here derandomization is less necessary, though sometimes still possible by using more sophisticated techniques.

We begin the chapter by looking at very simple randomized algorithms for two problems, the maximum satisfiability problem and the maximum cut problem. Here we show that sampling a solution uniformly at random from the set of all possible solutions gives a good randomized approximation algorithm. For the maximum satisfiability problem, we are able to go still further and show that biasing our choice yields a better performance guarantee. We then revisit the idea of using randomized rounding of linear programming relaxations introduced in Section 1.7, and show that it leads to still better approximation algorithms for the maximum satisfiability problem, as well as better algorithms for other problems we have seen previously, such as the prize-collecting Steiner tree problem, the uncapacitated facility location problem, and a single-machine scheduling problem.

We then give Chernoff bounds, which allow us to bound the probability that a sum of random variables is far away from its expected value. We show how these bounds can be applied to an integer multicommodity flow problem, which is historically the first use of randomized rounding. We end with a much more sophisticated use of drawing a random sample, and show that this technique can be used to 3-color certain kinds of dense 3-colorable graphs with high probability.

5.1 Simple Algorithms for MAX SAT and MAX CUT

Two problems will play an especially prominent role in our discussion of randomization in the design and analysis of approximation algorithms: the maximum satisfiability problem and the maximum cut problem. The former will be highlighted in this chapter, whereas the central developments for the latter will be deferred to the next chapter. However, in this section we will give a simple randomized $\frac{1}{2}$-approximation algorithm for each problem.

In the maximum satisfiability problem (often abbreviated as MAX SAT), the input consists of n Boolean variables $x_1, \ldots, x_n$ (each of which may be set to either true or false), m clauses $C_1, \ldots, C_m$ (each of which consists of a disjunction (that is, an "or") of some number of the variables and their negations – for example, $x_3 \vee \bar{x}_5 \vee x_{11}$, where $\bar{x}_i$ is the negation of x_i), and a nonnegative weight w_j for each clause C_j. The objective of the problem is to find an assignment of true/false to the x_i that maximizes the weight of the *satisfied* clauses. A clause is said to be satisfied if one of the unnegated variables is set to true, or one of the negated variables is set to false. For example, in the clause $x_3 \vee \bar{x}_5 \vee x_{11}$, the clause is not satisfied only if x_3 is set to false, x_5 to true, and x_{11} to false.

Some terminology will be useful in discussing the MAX SAT problem. We say that a variable x_i or a negated variable $\bar{x}_i$ is a *literal*, so that each clause consists of some number of literals. A variable x_i is called a *positive* literal, and a negated variable $\bar{x}_i$ is called a *negative* literal. The number of literals in a clause is called its *size* or *length*. We will denote the length of a clause C_j by l_j. Clauses of length one are sometimes called *unit* clauses. Without loss of generality, we assume that no literal is repeated in a clause (since this does not affect the satisfiability of the instance), and that at most one of x_i and $\bar{x}_i$ appears in a clause (since if both x_i and $\bar{x}_i$ are in a clause, it is trivially

satisfiable). Finally, it is natural to assume that the clauses are distinct, since we can simply sum the weights of two identical clauses.

A very straightforward use of randomization for MAX SAT is to set each x_i to true independently with probability $1/2$. An alternative perspective on this algorithm is that we choose a setting of the variables uniformly at random from the space of all possible settings. It turns out that this gives a reasonable approximation algorithm for this problem.

Theorem 5.1. *Setting each x_i to* true *with probability $1/2$ independently gives a randomized $\frac{1}{2}$-approximation algorithm for the maximum satisfiability problem.*

Proof. Consider a random variable Y_j such that Y_j is 1 if clause j is satisfied and 0 otherwise. Let W be a random variable that is equal to the total weight of the satisfied clauses, so that $W = \sum_{j=1}^{m} w_j Y_j$. Let OPT denote the optimum value of the MAX SAT instance. Then, by linearity of expectation, and the definition of the expectation of a 0-1 random variable, we know that

$$E[W] = \sum_{j=1}^{m} w_j E[Y_j] = \sum_{j=1}^{m} w_j \Pr[\text{clause } C_j \text{ satisfied}].$$

For each clause C_j, $j = 1, \ldots, n$, the probability that it is not satisfied is the probability that each positive literal in C_j is set to false and each negative literal in C_j is set to true, each of which happens with probability $1/2$ independently; hence,

$$\Pr[\text{clause } C_j \text{ satisfied}] = \left(1 - \left(\frac{1}{2}\right)^{l_j}\right) \geq \frac{1}{2},$$

where the last inequality is a consequence of the fact that $l_j \geq 1$. Hence,

$$E[W] \geq \frac{1}{2} \sum_{j=1}^{m} w_j \geq \frac{1}{2} \text{OPT},$$

where the last inequality follows from the fact that the total weight is an easy upper bound on the optimal value, since each weight is assumed to be nonnegative. □

Observe that if $l_j \geq k$ for each clause j, then the analysis above shows that the algorithm is a $\left(1 - \left(\frac{1}{2}\right)^k\right)$-approximation algorithm for such instances. Thus, the performance of the algorithm is better on MAX SAT instances consisting of long clauses. This observation will be useful to us later.

Although this seems like a pretty naive algorithm, a hardness theorem shows that this is the best that can be done in some cases. Consider the case in which $l_j = 3$ for all clauses j; this restriction of the problem is sometimes called MAX E3SAT, since there are exactly three literals in each clause. The analysis above shows that the randomized algorithm gives an approximation algorithm with performance guarantee $\left(1 - \left(\frac{1}{2}\right)^3\right) = \frac{7}{8}$. A truly remarkable result shows that nothing better is possible for these instances unless P = NP.

Theorem 5.2. *If there is a* $(\frac{7}{8} + \epsilon)$*-approximation algorithm for MAX E3SAT for any constant* $\epsilon > 0$*, then* P = NP.

We discuss this result further in Section 16.3.

In the maximum cut problem (sometimes abbreviated MAX CUT), the input is an undirected graph $G = (V, E)$, along with a nonnegative weight $w_{ij} \geq 0$ for each edge $(i, j) \in E$. The goal is to partition the vertex set into two parts, U and $W = V - U$, so as to maximize the weight of the edges whose two endpoints are in different parts, one in U and one in W. We say that an edge with endpoints in both U and W is *in the cut*. In the case $w_{ij} = 1$ for each edge $(i, j) \in E$, we have an *unweighted* MAX CUT problem.

It is easy to give a $\frac{1}{2}$-approximation algorithm for the MAX CUT problem along the same lines as the previous randomized algorithm for MAX SAT. Here we place each vertex $v \in V$ into U independently with probability $1/2$. As with the MAX SAT algorithm, this can be viewed as sampling a solution uniformly from the space of all possible solutions.

Theorem 5.3. *If we place each vertex* $v \in V$ *into* U *independently with probability* $1/2$*, then we obtain a randomized* $\frac{1}{2}$*-approximation algorithm for the maximum cut problem.*

Proof. Consider a random variable X_{ij} that is 1 if the edge (i, j) is in the cut, and 0 otherwise. Let Z be the random variable equal to the total weight of edges in the cut, so that $Z = \sum_{(i,j)\in E} w_{ij} X_{ij}$. Let OPT denote the optimal value of the maximum cut instance. Then, as before, by linearity of expectation and the definition of expectation of a 0-1 random variable, we get that

$$E[Z] = \sum_{(i,j)\in E} w_{ij} E[X_{ij}] = \sum_{(i,j)\in E} w_{ij} \Pr[\text{Edge } (i, j) \text{ in cut}].$$

In this case, the probability that a specific edge (i, j) is in the cut is easy to calculate: since the two endpoints are placed in the sets independently, they are in different sets with probability equal to $\frac{1}{2}$. Hence,

$$E[Z] = \frac{1}{2} \sum_{(i,j)\in E} w_{ij} \geq \frac{1}{2} \text{OPT},$$

where the inequality follows directly from the fact that the sum of the (nonnegative) weights of all edges is obviously an upper bound on the weight of the edges in an optimal cut. □

We will show in Section 6.2 that by using more sophisticated techniques we can get a substantially better performance guarantee for the MAX CUT problem.

5.2 Derandomization

As we mentioned in the introduction to the chapter, it is often possible to *derandomize* a randomized algorithm, that is, to obtain a deterministic algorithm whose solution value is as good as the expected value of the randomized algorithm.

To illustrate, we will show how the algorithm of the preceding section for the maximum satisfiability problem can be derandomized by replacing the randomized decision of whether to set x_i to true with a deterministic one that will preserve the expected value of the solution. These decisions will be made sequentially: the value of x_1 is determined first, then x_2, and so on.

How should x_1 be set so as to preserve the expected value of the algorithm? Assume for the moment that we will make the choice of x_1 deterministically, but all other variables will be set true with probability $1/2$ as before. Then the best way to set x_1 is that which will maximize the expected value of the resulting solution; that is, we should determine the expected value of W, the weight of satisfied clauses, given that x_1 is set to true, and the expected weight of W given that x_1 is set to false, and set x_1 to whichever value maximizes the expected value of W. It makes intuitive sense that this should work, since the maximum is always greater than an average, and the expected value of W is the average of its expected value given the two possible settings of x_1. In this way, we maintain an algorithmic invariant that the expected value is at least half the optimum, while having fewer random variables left.

More formally, if $E[W|x_1 \leftarrow \text{true}] \geq E[W|x_1 \leftarrow \text{false}]$, then we set x_1 true; otherwise, we set it to false. Since by the definition of conditional expectations,

$$\begin{aligned} E[W] &= E[W|x_1 \leftarrow \text{true}]\Pr[x_1 \leftarrow \text{true}] + E[W|x_1 \leftarrow \text{false}]\Pr[x_1 \leftarrow \text{false}] \\ &= \frac{1}{2}\left(E[W|x_1 \leftarrow \text{true}] + E[W|x_1 \leftarrow \text{false}]\right), \end{aligned}$$

if we set x_1 to truth value b_1 so as to maximize the conditional expectation, then $E[W|x_1 \leftarrow b_1] \geq E[W]$; that is, the deterministic choice of how to set x_1 guarantees an expected value no less than the expected value of the completely randomized algorithm.

Assuming for the moment that we can compute these conditional expectations, the deterministic decision of how to set the remaining variables is similar. Assume that we have set variables $x_1, \ldots, x_i$ to truth values $b_1, \ldots, b_i$, respectively. How shall we set variable x_{i+1}? Again, assume that the remaining variables are set randomly. Then the best way to set x_{i+1} is so as to maximize the expected value given the previous settings of $x_1, \ldots, x_i$. So if $E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i, x_{i+1} \leftarrow \text{true}] \geq E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i, x_{i+1} \leftarrow \text{false}]$, we set x_{i+1} to true (thus setting b_{i+1} to true); otherwise, we set x_{i+1} to false (thus setting b_{i+1} to false). Then since

$$\begin{aligned} &E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i] \\ &\quad = E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i, x_{i+1} \leftarrow \text{true}]\Pr[x_{i+1} \leftarrow \text{true}] \\ &\qquad + E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i, x_{i+1} \leftarrow \text{false}]\Pr[x_{i+1} \leftarrow \text{false}] \\ &\quad = \frac{1}{2}(E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i, x_{i+1} \leftarrow \text{true}] \\ &\qquad + E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i, x_{i+1} \leftarrow \text{false}]), \end{aligned}$$

setting x_{i+1} to truth value b_{i+1} as described above ensures that

$$E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i, x_{i+1} \leftarrow b_{i+1}] \geq E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i].$$

By induction, this implies that $E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i, x_{i+1} \leftarrow b_{i+1}] \geq E[W]$.

We continue this process until all n variables have been set. Then since the conditional expectation given the setting of all n variables, $E[W|x_1 \leftarrow b_1, \ldots, x_n \leftarrow b_n]$, is simply the value of the solution given by the deterministic algorithm, we know that the value of the solution returned is at least $E[W] \geq \frac{1}{2}\,\mathrm{OPT}$. Therefore, the algorithm is a $\frac{1}{2}$-approximation algorithm.

These conditional expectations are not difficult to compute. By definition,

$$\begin{aligned} E[W|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i] &= \sum_{j=1}^{m} w_j E[Y_j|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i] \\ &= \sum_{j=1}^{m} w_j \Pr[\text{clause } C_j \text{ satisfied}|x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i]. \end{aligned}$$

Furthermore, the probability that clause C_j is satisfied given that $x_1 \leftarrow b_1, \ldots, x_i \leftarrow b_i$ is easily seen to be 1 if the settings of $x_1, \ldots, x_i$ already satisfy the clause, and is $1 - (1/2)^k$ otherwise, where k is the number of literals in the clause that remain unset by this procedure. For example, consider the clause $x_3 \vee \bar{x}_5 \vee \bar{x}_7$. It is the case that

$$\Pr[\text{clause satisfied}|x_1 \leftarrow \text{true}, x_2 \leftarrow \text{false}, x_3 \leftarrow \text{true}] = 1,$$

since setting x_3 to true satisfies the clause. On the other hand,

$$\Pr[\text{clause satisfied}|x_1 \leftarrow \text{true}, x_2 \leftarrow \text{false}, x_3 \leftarrow \text{false}] = 1 - \left(\frac{1}{2}\right)^2 = \frac{3}{4},$$

since the clause will be unsatisfied only if x_5 and x_7 are set true, an event that occurs with probability 1/4.

This technique for derandomizing algorithms works with a wide variety of randomized algorithms in which variables are set independently and the conditional expectations are polynomial-time computable. It is sometimes called the *method of conditional expectations*, due to its use of conditional expectations. In particular, an almost identical argument leads to a derandomized version of the randomized $\frac{1}{2}$-approximation algorithm for the MAX CUT problem. Most of the randomized algorithms we discuss in this chapter can be derandomized via this method. The randomized versions of the algorithms are easier to present and analyze, and so we will frequently not discuss their deterministic variants.

5.3 Flipping Biased Coins

How might we improve the randomized algorithm for MAX SAT? We will show here that biasing the probability with which we set x_i is actually helpful; that is, we will set x_i true with some probability not equal to $1/2$. To do this, it is easiest to start by considering only MAX SAT instances with no unit clauses $\bar{x}_i$, that is, no negated unit

clauses. We will later show that we can remove this assumption. Suppose now we set each x_i to be true independently with probability $p > 1/2$. As in the analysis of the previous randomized algorithm, we will need to analyze the probability that any given clause is satisfied.

Lemma 5.4. *If each x_i is set to* true *with probability $p > 1/2$ independently, then the probability that any given clause is satisfied is at least* $\min(p, 1 - p^2)$ *for MAX SAT instances with no negated unit clauses.*

Proof. If the clause is a unit clause, then the probability the clause is satisfied is p, since it must be of the form x_i, and the probability x_i is set true is p. If the clause has length at least two, then the probability that the clause is satisfied is $1 - p^a(1 - p)^b$, where a is the number of negated variables in the clause and b is the number of unnegated variables in the clause, so that $a + b = l_j \geq 2$. Since $p > \frac{1}{2} > 1 - p$, this probability is at least $1 - p^{a+b} = 1 - p^{l_j} \geq 1 - p^2$, and the lemma is proved. □

We can obtain the best performance guarantee by setting $p = 1 - p^2$. This yields $p = \frac{1}{2}(\sqrt{5} - 1) \approx 0.618$. The lemma immediately implies the following theorem.

Theorem 5.5. *Setting each x_i to* true *with probability p independently gives a randomized* $\min(p, 1 - p^2)$*-approximation algorithm for MAX SAT instances with no negated unit clauses.*

Proof. This follows since

$$E[W] = \sum_{j=1}^{m} w_j \Pr[\text{clause } C_j \text{ satisfied}] \geq \min(p, 1 - p^2) \sum_{j=1}^{m} w_j$$
$$\geq \min(p, 1 - p^2)\,\mathrm{OPT}\,.$$

□

We would like to extend this result to all MAX SAT instances. To do this, we will use a better bound on OPT than $\sum_{j=1}^{m} w_j$. Assume that for every i the weight of the unit clause x_i appearing in the instance is at least the weight of the unit clause $\bar{x}_i$; this is without loss of generality since we could negate all occurrences of x_i if the assumption is not true. Let v_i be the weight of the unit clause $\bar{x}_i$ if it exists in the instance, and let v_i be 0 otherwise.

Lemma 5.6. $\mathrm{OPT} \leq \sum_{j=1}^{m} w_j - \sum_{i=1}^{n} v_i$.

Proof. For each i, the optimal solution can satisfy exactly one of x_i and $\bar{x}_i$. Thus, the weight of the optimal solution cannot include both the weight of the clause x_i and the clause $\bar{x}_i$. Since v_i is the smaller of these two weights, the lemma follows. □

We can now extend the result.

Theorem 5.7. *We can obtain a randomized* $\frac{1}{2}(\sqrt{5} - 1)$*-approximation algorithm for MAX SAT.*

Proof. Let U be the set of indices of clauses of the instance excluding unit clauses of the form $\bar{x}_i$. As above, we assume without loss of generality that the weight of each clause

$\bar{x}_i$ is no greater than the weight of clause x_i. Thus, $\sum_{j \in U} w_j = \sum_{j=1}^{m} w_j - \sum_{i=1}^{n} v_i$. Then set each x_i to be true independently with probability $p = \frac{1}{2}(\sqrt{5} - 1)$. Then

$$\begin{aligned} E[W] &= \sum_{j=1}^{m} w_j \Pr[\text{clause } C_j \text{ satisfied}] \\ &\geq \sum_{j \in U} w_j \Pr[\text{clause } C_j \text{ satisfied}] \\ &\geq p \cdot \sum_{j \in U} w_j \qquad (5.1) \\ &= p \cdot \left(\sum_{j=1}^{m} w_j - \sum_{i=1}^{n} v_i \right) \geq p \cdot \text{OPT}, \end{aligned}$$

where (5.1) follows by Theorem 5.5 and the fact that $p = \min(p, 1 - p^2)$. □

This algorithm can be derandomized using the method of conditional expectations.

5.4 Randomized Rounding

The algorithm of the previous section shows that biasing the probability with which we set x_i true yields an improved approximation algorithm. However, we gave each variable the same bias. In this section, we show that we can do still better by giving each variable its own bias. We do this by returning to the idea of *randomized rounding*, which we examined briefly in Section 1.7 in the context of the set cover problem.

Recall that in randomized rounding, we first set up an integer programming formulation of the problem at hand in which there are 0-1 integer variables. In this case we will create an integer program with a 0-1 variable y_i for each Boolean variable x_i such that $y_i = 1$ corresponds to x_i set true. The integer program is relaxed to a linear program by replacing the constraints $y_i \in \{0, 1\}$ with $0 \leq y_i \leq 1$, and the linear programming relaxation is solved in polynomial time. Recall that the central idea of randomized rounding is that the fractional value y_i^* is interpreted as the probability that y_i should be set to 1. In this case, we set each x_i to true with probability y_i^* independently.

We now give an integer programming formulation of the MAX SAT problem. In addition to the variables y_i, we introduce a variable z_j for each clause C_j that will be 1 if the clause is satisfied and 0 otherwise. For each clause C_j, let P_j be the indices of the variables x_i that occur positively in the clause, and let N_j be the indices of the variables x_i that are negated in the clause. We denote the clause C_j by

$$\bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \bar{x}_i.$$

Then the inequality

$$\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j$$

must hold for clause C_j since if each variable that occurs positively in the clause is set to false (and its corresponding y_i is set to 0) and each variable that occurs negatively is set to true (and its corresponding y_i is set to 1), then the clause is not satisfied, and z_j must be 0. This inequality yields the following integer programming formulation of the MAX SAT problem:

$$\begin{aligned}
&\text{maximize} \sum_{j=1}^{m} w_j z_j \\
&\text{subject to} \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j, \qquad \forall C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \bar{x}_i, \\
&\qquad\qquad y_i \in \{0, 1\}, \qquad i = 1, \ldots, n, \\
&\qquad\qquad 0 \leq z_j \leq 1, \qquad j = 1, \ldots, m.
\end{aligned}$$

If Z^*_{IP} is the optimal value of this integer program, then it is not hard to see that $Z^*_{IP} = \text{OPT}$.

The corresponding linear programming relaxation of this integer program is

$$\begin{aligned}
&\text{maximize} \sum_{j=1}^{m} w_j z_j \\
&\text{subject to} \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j, \qquad \forall C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \bar{x}_i, \\
&\qquad\qquad 0 \leq y_i \leq 1, \qquad i = 1, \ldots, n, \\
&\qquad\qquad 0 \leq z_j \leq 1, \qquad j = 1, \ldots, m.
\end{aligned}$$

If Z^*_{LP} is the optimal value of this linear program, then clearly $Z^*_{LP} \geq Z^*_{IP} = \text{OPT}$.

Let (y^*, z^*) be an optimal solution to the linear programming relaxation. We now consider the result of using randomized rounding, and setting x_i to true with probability y^*_i independently. Before we can begin the analysis, we will need two facts. The first is commonly called the *arithmetic-geometric mean inequality* because it compares the arithmetic and geometric means of a set of numbers.

Fact 5.8 (Arithmetic-geometric mean inequality). *For any nonnegative $a_1, \ldots, a_k$,*

$$\left(\prod_{i=1}^{k} a_i \right)^{1/k} \leq \frac{1}{k} \sum_{i=1}^{k} a_i.$$

Fact 5.9. *If a function $f(x)$ is concave on the interval $[0, 1]$ (that is, $f''(x) \leq 0$ on $[0, 1]$), and $f(0) = a$ and $f(1) = b + a$, then $f(x) \geq bx + a$ for $x \in [0, 1]$ (see Figure 5.1).*

Theorem 5.10. *Randomized rounding gives a randomized $(1 - \frac{1}{e})$-approximation algorithm for MAX SAT.*

Proof. As in the analyses of the algorithms in the previous sections, the main difficulty is analyzing the probability that a given clause C_j is satisfied. Pick an arbitrary clause

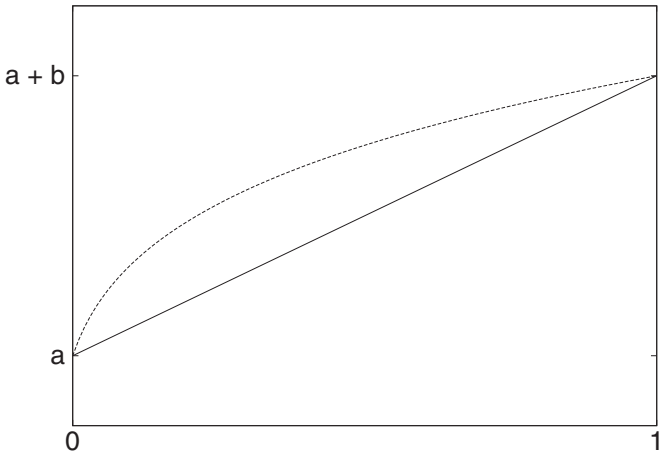

Figure 5.1. An illustration of Fact 5.9.

C_j. Then, by applying the arithmetic-geometric mean inequality, we see that

$$\Pr[\text{clause } C_j \text{ not satisfied}] = \prod_{i \in P_j}(1 - y_i^*) \prod_{i \in N_j} y_i^* \leq \left[\frac{1}{l_j} \left(\sum_{i \in P_j}(1 - y_i^*) + \sum_{i \in N_j} y_i^* \right) \right]^{l_j}.$$

By rearranging terms, we can derive that

$$\left[\frac{1}{l_j} \left(\sum_{i \in P_j}(1 - y_i^*) + \sum_{i \in N_j} y_i^* \right) \right]^{l_j} = \left[1 - \frac{1}{l_j} \left(\sum_{i \in P_j} y_i^* + \sum_{i \in N_j}(1 - y_i^*) \right) \right]^{l_j}.$$

By invoking the corresponding inequality from the linear program,

$$\sum_{i \in P_j} y_i^* + \sum_{i \in N_j}(1 - y_i^*) \geq z_j^*,$$

we see that

$$\Pr[\text{clause } C_j \text{ not satisfied}] \leq \left(1 - \frac{z_j^*}{l_j}\right)^{l_j}.$$

The function $f(z_j^*) = 1 - \left(1 - \frac{z_j^*}{l_j}\right)^{l_j}$ is concave for $l_j \geq 1$. Then by using Fact 5.9,

$$\begin{aligned} \Pr[\text{clause } C_j \text{ satisfied}] &\geq 1 - \left(1 - \frac{z_j^*}{l_j}\right)^{l_j} \\ &\geq \left[1 - \left(1 - \frac{1}{l_j}\right)^{l_j} \right] z_j^*. \end{aligned}$$

Therefore, the expected value of the randomized rounding algorithm is

$$\begin{aligned} E[W] &= \sum_{j=1}^{m} w_j \Pr[\text{clause } C_j \text{ satisfied}] \\ &\geq \sum_{j=1}^{m} w_j z_j^* \left[1 - \left(1 - \frac{1}{l_j}\right)^{l_j}\right] \\ &\geq \min_{k \geq 1} \left[1 - \left(1 - \frac{1}{k}\right)^{k}\right] \sum_{j=1}^{m} w_j z_j^*. \end{aligned}$$

Note that $\left[1 - \left(1 - \frac{1}{k}\right)^k\right]$ is a non-increasing function in k and that it approaches $\left(1 - \frac{1}{e}\right)$ from above as k tends to infinity. Since $\sum_{j=1}^{m} w_j z_j^* = Z_{LP}^* \geq \text{OPT}$, we have that

$$E[W] \geq \min_{k \geq 1} \left[1 - \left(1 - \frac{1}{k}\right)^{k}\right] \sum_{j=1}^{m} w_j z_j^* \geq \left(1 - \frac{1}{e}\right) \text{OPT}. \qquad \square$$

This randomized rounding algorithm can be derandomized in the standard way using the method of conditional expectations.

5.5 Choosing the Better of Two Solutions

In this section we observe that choosing the better solution from the two given by the randomized rounding algorithm of the previous section and the unbiased randomized algorithm of the first section gives a better performance guarantee than that of either algorithm. This happens because, as we shall see, the algorithms have contrasting bad cases: when one algorithm is far from optimal, the other is close, and vice versa. This technique can be useful in other situations, and does not require using randomized algorithms.

In this case, consider a given clause C_j of length l_j. The randomized rounding algorithm of Section 5.4 satisfies the clause with probability at least $\left[1 - \left(1 - \frac{1}{l_j}\right)^{l_j}\right] z_j^*$, while the unbiased randomized algorithm of Section 5.1 satisfies the clause with probability $1 - 2^{-l_j} \geq (1 - 2^{-l_j}) z_j^*$. Thus, when the clause is short, it is very likely to be satisfied by the randomized rounding algorithm, though not by the unbiased randomized algorithm, and when the clause is long the opposite is true. This observation is made precise and rigorous in the following theorem.

Theorem 5.11. *Choosing the better of the two solutions given by the randomized rounding algorithm and the unbiased randomized algorithm yields a randomized $\frac{3}{4}$-approximation algorithm for MAX SAT.*

Proof. Let W_1 be a random variable denoting the value of the solution returned by the randomized rounding algorithm, and let W_2 be a random variable denoting the value

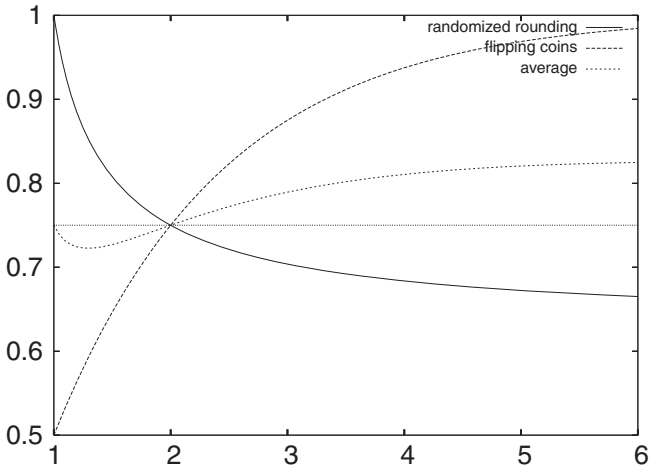

Figure 5.2. Illustration of the proof of Theorem 5.11. The "randomized rounding" line is the function $1-(1-\frac{1}{k})^k$. The "flipping coins" line is the function $1-2^{-k}$. The "average" line is the average of these two functions, which is at least $\frac{3}{4}$ for all integers $k \geq 1$.

of the solution returned by the unbiased randomized algorithm. Then we wish to show that

$$E[\max(W_1, W_2)] \geq \frac{3}{4}\,\mathrm{OPT}\,.$$

To obtain this inequality, observe that

$$\begin{aligned}
E[\max(W_1, W_2)] &\geq E\left[\frac{1}{2}W_1 + \frac{1}{2}W_2\right] \\
&= \frac{1}{2}E[W_1] + \frac{1}{2}E[W_2] \\
&\geq \frac{1}{2}\sum_{j=1}^{m} w_j z_j^* \left[1 - \left(1 - \frac{1}{l_j}\right)^{l_j}\right] + \frac{1}{2}\sum_{j=1}^{m} w_j \left(1 - 2^{-l_j}\right) \\
&\geq \sum_{j=1}^{m} w_j z_j^* \left[\frac{1}{2}\left(1 - \left(1 - \frac{1}{l_j}\right)^{l_j}\right) + \frac{1}{2}\left(1 - 2^{-l_j}\right)\right].
\end{aligned}$$

We claim that

$$\left[\frac{1}{2}\left(1 - \left(1 - \frac{1}{l_j}\right)^{l_j}\right) + \frac{1}{2}\left(1 - 2^{-l_j}\right)\right] \geq \frac{3}{4}$$

for all positive integers l_j. We will prove this shortly, but this can be seen in Figure 5.2. Given the claim, we have that

$$E[\max(W_1, W_2)] \geq \frac{3}{4}\sum_{j=1}^{m} w_j z_j^* = \frac{3}{4} Z_{LP}^* \geq \frac{3}{4}\,\mathrm{OPT}\,.$$

Now to prove the claim. Observe that the claim holds for $l_j = 1$, since

$$\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4},$$

and the claim holds for $l_j = 2$, since

$$\frac{1}{2} \cdot \left(1 - \left(\frac{1}{2}\right)^2\right) + \frac{1}{2}(1 - 2^{-2}) = \frac{3}{4}.$$

For all $l_j \geq 3$, $\left(1 - \left(1 - \frac{1}{l_j}\right)^{l_j}\right) \geq 1 - \frac{1}{e}$ and $\left(1 - 2^{-l_j}\right) \geq \frac{7}{8}$, and

$$\frac{1}{2}\left(1 - \frac{1}{e}\right) + \frac{1}{2} \cdot \frac{7}{8} \approx 0.753 \geq \frac{3}{4},$$

so the claim is proven. □

Notice that taking the better solution of the two derandomized algorithms gives at least $\max(E[W_1], E[W_2]) \geq \frac{1}{2}E[W_1] + \frac{1}{2}E[W_2]$. The proof above shows that this quantity is at least $\frac{3}{4}$ OPT. Thus, taking the better solution of the two derandomized algorithms is a deterministic $\frac{3}{4}$-approximation algorithm.

5.6 Nonlinear Randomized Rounding

Thus far in our applications of randomized rounding, we have used the variable y_i^* from the linear programming relaxation as a probability to decide whether to set y_i to 1 in the integer programming formulation of the problem. In the case of the MAX SAT problem, we set x_i to true with probability y_i^*. There is no reason, however, that we cannot use some function $f : [0, 1] \to [0, 1]$ to set x_i to true with probability $f(y_i^*)$. Sometimes this yields approximation algorithms with better performance guarantees than using the identity function, as we will see in this section.

In this section we will show that a $\frac{3}{4}$-approximation algorithm for MAX SAT can be obtained directly by using randomized rounding with a nonlinear function f. In fact, there is considerable freedom in choosing such a function f: let f be any function such that $f : [0, 1] \to [0, 1]$ and

$$1 - 4^{-x} \leq f(x) \leq 4^{x-1}. \tag{5.2}$$

See Figure 5.3 for a plot of the bounding functions. We will see that this ensures that the probability a clause C_j is satisfied is at least $1 - 4^{-z_j^*} \geq \frac{3}{4}z_j^*$, which will give the $\frac{3}{4}$-approximation algorithm.

Theorem 5.12. *Randomized rounding with the function f is a randomized $\frac{3}{4}$-approximation algorithm for MAX SAT.*

Proof. Once again, it suffices to analyze the probability that a given clause C_j is satisfied. By the definition of f,

$$\Pr[\text{clause } C_j \text{ not satisfied}] = \prod_{i \in P_j}(1 - f(y_i^*)) \prod_{i \in N_j} f(y_i^*) \leq \prod_{i \in P_j} 4^{-y_i^*} \prod_{i \in N_j} 4^{y_i^* - 1}.$$

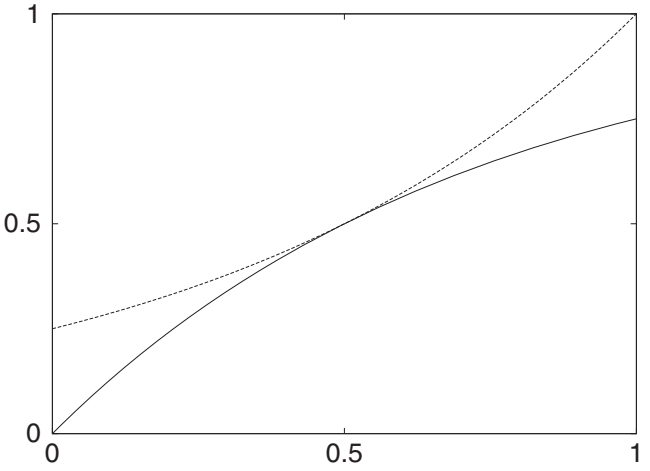

Figure 5.3. A plot of the functions of (5.2). The upper curve is 4^{x-1} and the lower curve is $1 - 4^{-x}$.

Rewriting the product and using the linear programming constraint for clause C_j gives us

$$\Pr[\text{clause } C_j \text{ not satisfied}] \leq \prod_{i \in P_j} 4^{-y_i^*} \prod_{i \in N_j} 4^{y_i^* - 1} = 4^{-\left(\sum_{i \in P_j} y_i^* + \sum_{i \in N_j} (1 - y_i^*)\right)} \leq 4^{-z_j^*}.$$

Then using Fact 5.9 and observing that the function $g(z) = 1 - 4^{-z}$ is concave on [0,1], we have

$$\Pr[\text{clause } C_j \text{ satisfied}] \geq 1 - 4^{-z_j^*} \geq (1 - 4^{-1}) z_j^* = \frac{3}{4} z_j^*.$$

It follows that the expected performance of the algorithm is

$$E[W] = \sum_{j=1}^{m} w_j \Pr[\text{clause } C_j \text{ satisfied}] \geq \frac{3}{4} \sum_{j=1}^{m} w_j z_j^* \geq \frac{3}{4} \text{OPT}. \qquad \square$$

Once again, the algorithm can be derandomized using the method of conditional expectations.

There are other choices of the function f that also lead to a $\frac{3}{4}$-approximation algorithm for MAX SAT. Some other possibilities are presented in the exercises at the end of the chapter.

Is it possible to get an algorithm with a performance guarantee better than $\frac{3}{4}$ by using some more complicated form of randomized rounding? It turns out that the answer is no, at least for any algorithm that derives its performance guarantee by comparing its value to that of the linear programming relaxation. To see this, consider the instance of the maximum satisfiability problem with two variables x_1 and x_2 and four clauses of weight 1 each, $x_1 \vee x_2$, $x_1 \vee \bar{x}_2$, $\bar{x}_1 \vee x_2$, and $\bar{x}_1 \vee \bar{x}_2$. Any feasible solution, including the optimal solution, satisfies exactly three of the four clauses. However, if we set $y_1 = y_2 = \frac{1}{2}$ and $z_j = 1$ for all four clauses C_j, then this solution is feasible for the linear program and has value 4. Thus, the value to any solution to the MAX SAT instance can be at most $\frac{3}{4} \sum_{j=1}^{m} w_j z_j^*$. We say that this integer programming formulation of the maximum satisfiability problem has an *integrality gap* of $\frac{3}{4}$.

Definition 5.13. *The* integrality gap *of an integer program is the worst-case ratio over all instances of the problem of the value of an optimal solution to the integer programming formulation to the value of an optimal solution to its linear programming relaxation.*

The example above shows that the integrality gap is at most $\frac{3}{4}$, whereas the proof of Theorem 5.12 shows that it is at least $\frac{3}{4}$. If an integer programming formulation has integrality gap ρ, then any algorithm for a maximization problem whose performance guarantee α is proven by showing that the value of its solution is at least α times the value of the linear programming relaxation can have performance guarantee at most ρ. A similar statement holds for minimization problems.

5.7 The Prize-Collecting Steiner Tree Problem

We now consider other problems for which randomized techniques are useful. In particular, in the next few sections, we revisit some of the problems we studied earlier in the book and show that we can obtain improved performance guarantees by using the randomized methods that we have developed in this chapter.

We start by returning to the prize-collecting Steiner tree problem we discussed in Section 4.4. Recall that in this problem we are given an undirected graph $G = (V, E)$, an edge cost $c_e \geq 0$ for each $e \in E$, a selected root vertex $r \in V$, and a penalty $\pi_i \geq 0$ for each $i \in V$. The goal is to find a tree T that contains the root vertex r so as to minimize $\sum_{e \in T} c_e + \sum_{i \in V - V(T)} \pi_i$, where $V(T)$ is the set of vertices in the tree. We used the following linear programming relaxation of the problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e x_e + \sum_{i \in V} \pi_i (1 - y_i) \\
\text{subject to} \quad & \sum_{e \in \delta(S)} x_e \geq y_i, \qquad \forall S \subseteq V - r, S \neq \emptyset, \forall i \in S, \\
& y_r = 1, \\
& y_i \geq 0, \qquad \forall i \in V, \\
& x_e \geq 0, \qquad \forall e \in E.
\end{aligned}$$

In the 3-approximation algorithm given in Section 4.4, we found an optimal LP solution (x^*, y^*), and for a specified value of α, we built a Steiner tree on all nodes such that $y_i^* \geq \alpha$. We claimed that the cost of the edges in the tree is within a factor of $2/\alpha$ of the fractional cost of the tree edges in the LP solution, while the cost of the penalties is within a factor of $1/(1 - \alpha)$ of the cost of the penalties in the LP solution. Thus, if α is close to 1, the cost of the tree edges is within a factor of 2 of the corresponding cost of the LP, while if α is close to 0, our penalty cost is close to the corresponding cost of the LP.

In the previous section we set $\alpha = 2/3$ to trade off the costs of the tree edges and the penalties, resulting in a performance guarantee of 3. Suppose instead that we choose the value of α randomly rather than considering just one value of α. We will see that this improves the performance guarantee from 3 to about 2.54.

Recall Lemma 4.6 from Section 4.4, which allowed us to bound the cost of the spanning tree T constructed in terms of α and the LP solution (x^*, y^*).

Lemma 5.14 (Lemma 4.6).

$$\sum_{e \in T} c_e \leq \frac{2}{\alpha} \sum_{e \in E} c_e x_e^*.$$

Because the bound becomes infinite as α tends to zero, we don't want to choose α too close to zero. Instead, we will choose α uniformly from the range $[\gamma, 1]$, and will later decide how to set γ. Recall that in computing the expected value of a continuous random variable X, if that variable has a probability density function $f(x)$ over the domain $[a, b]$ (specifying the probability that $X = x$), then we compute the expectation of X by integrating $f(x)xdx$ over that interval. The probability density function for a uniform random variable over $[\gamma, 1]$ is the constant $1/(1 - \gamma)$. We can then analyze the expected cost of the tree for the randomized algorithm below.

Lemma 5.15.

$$E\left[\sum_{e \in T} c_e\right] \leq \left(\frac{2}{1-\gamma} \ln \frac{1}{\gamma}\right) \sum_{e \in E} c_e x_e^*.$$

Proof. Using simple algebra and calculus, we obtain

$$\begin{aligned} E\left[\sum_{e \in T} c_e\right] &\leq E\left[\frac{2}{\alpha} \sum_{e \in E} c_e x_e^*\right] \\ &= E\left[\frac{2}{\alpha}\right] \sum_{e \in E} c_e x_e^* \\ &= \left(\frac{1}{1-\gamma} \int_\gamma^1 \frac{2}{x} dx\right) \sum_{e \in E} c_e x_e^* \\ &= \left[\frac{2}{1-\gamma} \ln x\right]_\gamma^1 \cdot \sum_{e \in E} c_e x_e^* \\ &= \left(\frac{2}{1-\gamma} \ln \frac{1}{\gamma}\right) \sum_{e \in E} c_e x_e^*. \end{aligned}$$

□

The expected penalty for the vertices not in the tree is also easy to analyze.

Lemma 5.16.

$$E\left[\sum_{i \in V - V(T)} \pi_i\right] \leq \frac{1}{1-\gamma} \sum_{i \in V} \pi_i (1 - y_i^*)$$

Proof. Let $U = \left\{i \in V : y_i^* \geq \alpha\right\}$; any vertex not in the tree must not be in U, so we have that $\sum_{i \in V - V(T)} \pi_i \leq \sum_{i \notin U} \pi_i$. Observe that if $y_i^* \geq \gamma$, then the probability that $i \notin U$ is $(1 - y_i^*)/(1 - \gamma)$. If $y_i^* < \gamma$, then $i \notin U$ with probability 1. But then $1 \leq (1 - y_i^*)/(1 - \gamma)$, and so the lemma statement follows. □

Thus, we have the following theorem and corollary.

Theorem 5.17. *The expected cost of the solution produced by the randomized algorithm is*

$$E\left[\sum_{e\in T} c_e + \sum_{i\in V-V(T)} \pi_i\right] \le \left(\frac{2}{1-\gamma}\ln\frac{1}{\gamma}\right)\sum_{e\in E} c_e x_e^* + \frac{1}{1-\gamma}\sum_{i\in V}\pi_i(1-y_i^*).$$

Corollary 5.18. *Using the randomized rounding algorithm with $\gamma = e^{-1/2}$ gives a $(1-e^{-1/2})^{-1}$-approximation algorithm for the prize-collecting Steiner tree problem, where $(1-e^{-1/2})^{-1} \approx 2.54$.*

Proof. We want to ensure that the maximum of the coefficients of the two terms in the bound of Theorem 5.17 is as small as possible. The first coefficient is a decreasing function of γ, and the second is an increasing function. We can minimize the maximum by setting them equal. By setting $\frac{2}{1-\gamma}\ln\frac{1}{\gamma} = \frac{1}{1-\gamma}$, we obtain $\gamma = e^{-1/2}$. Then by Theorem 5.17, the expected cost of the tree obtained is no greater than

$$\frac{1}{1-e^{-1/2}}\left(\sum_{e\in E} c_e x_e^* + \sum_{i\in V}\pi_i(1-y_i^*)\right) \le \frac{1}{1-e^{-1/2}}\cdot \mathrm{OPT}\,. \qquad \square$$

The derandomization of the algorithm is straightforward: since there are $|V|$ variables y_i^*, there are at most $|V|$ distinct values of y_i^*. Thus, consider the $|V|$ sets $U_j = \{i \in V : y_i^* \ge y_j^*\}$. Any possible value of α corresponds to one of these $|V|$ sets. Thus, if a random choice of α has a certain expected performance guarantee, the algorithm that tries each set U_j and chooses the best solution generated will have a deterministic performance guarantee at least as good as that of the expectation of the randomized algorithm. Interestingly, the use of randomization allows us to analyze this natural deterministic algorithm, whereas we know of no means of analyzing the deterministic algorithm directly.

Recall from the end of the previous section that we defined the integrality gap of an integer programming formulation to be the worst-case ratio, over all instances of a problem, of the value of an optimal solution to the integer programming formulation to the value of an optimal solution to the linear programming relaxation. We also explained that the integrality gap bounds the performance guarantee that we can get via LP rounding arguments. Consider a graph G that is a cycle on n nodes, and let the penalty for each node be infinite and the cost of each edge be 1. Then there is a feasible solution to the linear programming relaxation of cost $n/2$ by setting each edge variable to $1/2$, while there is an optimal integral solution of cost $n-1$ by taking every edge of the cycle except one (see Figure 5.4). Hence, the integrality gap for this instance of the problem is at least $(n-1)/(n/2) = 2 - 2/n$. Thus, we cannot expect a performance guarantee for the prize-collecting Steiner tree problem better than $2 - \frac{2}{n}$ using LP rounding arguments with this formulation. In Chapter 14, we will return to the prize-collecting Steiner tree problem and show that the primal-dual method can be used to obtain a 2-approximation algorithm for the problem. The argument there will show that the integrality gap of the integer programming formulation is at most 2.

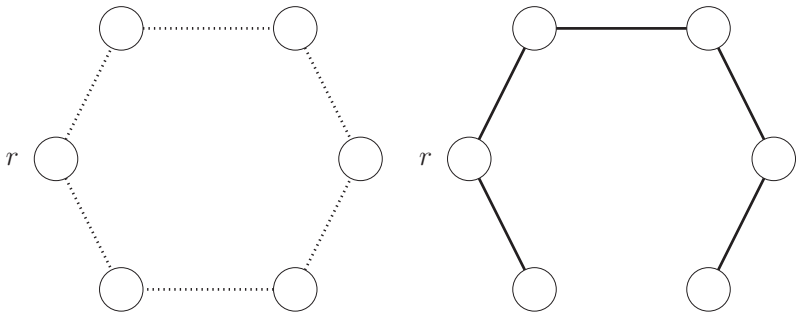

Figure 5.4. Example of the integrality gap of the integer programming formulation for the prize-collecting Steiner tree problem. On the left is a feasible solution for the linear program in which each edge has value 1/2. On the right is an optimal solution for the integer programming formulation in which each edge shown has value 1.

5.8 The Uncapacitated Facility Location Problem

In this section, we revisit the metric uncapacitated facility location problem introduced in Section 4.5. Recall that in this problem we are given a set of clients D and a set of facilities F, along with facility costs f_i for all facilities $i \in F$, and assignment costs c_{ij} for all facilities $i \in F$ and clients $j \in D$. All clients and facilities are points in a metric space, and given clients j, l and facilities i, k, we have that $c_{ij} \leq c_{il} + c_{kl} + c_{kj}$. The goal of the problem is to select a subset of facilities to open and an assignment of clients to open facilities so as to minimize the total cost of the open facilities plus the assignment costs. We used the following linear programming relaxation of the problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in F} f_i y_i + \sum_{i \in F, j \in D} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{i \in F} x_{ij} = 1, \qquad \forall j \in D, \\
& x_{ij} \leq y_i, \qquad \forall i \in F, j \in D, \\
& x_{ij} \geq 0, \qquad \forall i \in F, j \in D, \\
& y_i \geq 0, \qquad \forall i \in F,
\end{aligned}$$

where the variable x_{ij} indicates whether client j is assigned to facility i, and the variable y_i indicates whether facility i is open or not. We also used the dual of the LP relaxation:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j \in D} v_j \\
\text{subject to} \quad & \sum_{j \in D} w_{ij} \leq f_i, \qquad \forall i \in F, \\
& v_j - w_{ij} \leq c_{ij}, \qquad \forall i \in F, j \in D, \\
& w_{ij} \geq 0, \qquad \forall i \in F, j \in D.
\end{aligned}$$

Solve LP, get optimal primal solution (x^*, y^*) and dual solution (v^*, w^*)
$C \leftarrow D$
$k \leftarrow 0$
while $C \neq \emptyset$ **do**
  $k \leftarrow k + 1$
  Choose $j_k \in C$ that minimizes $v_j^* + C_j^*$ over all $j \in C$
  Choose $i_k \in N(j_k)$ according to the probability distribution $x_{ij_k}^*$
  Assign j_k and all unassigned clients in $N^2(j_k)$ to i_k
  $C \leftarrow C - \{j_k\} - N^2(j_k)$

Algorithm 5.1. Randomized rounding algorithm for the uncapacitated facility location problem.

Finally, given an LP solution (x^*, y^*), we said that a client j neighbors a facility i if $x_{ij}^* > 0$. We denote the neighbors of j as $N(j) = \{i \in F : x_{ij}^* > 0\}$, and the neighbors of the neighbors of j as $N^2(j) = \{k \in D : \exists i \in N(j), x_{ik}^* > 0\}$. Recall that we showed in Lemma 4.11 that if (v^*, w^*) is an optimal dual solution and $i \in N(j)$ then the cost of assigning j to i is bounded by v_j^* (that is, $c_{ij} \leq v_j^*$).

We gave a 4-approximation algorithm in Algorithm 4.2 of Section 4.5 that works by choosing an unassigned client j that minimizes the value of v_j^* among all remaining unassigned clients, opening the cheapest facility in the neighborhood of $N(j)$, and then assigning j and all clients in $N^2(j)$ to this facility. We showed that for an optimal LP solution (x^*, y^*) and optimal dual solution v^*, this gave a solution of cost at most $\sum_{i \in F} f_i y_i^* + 3 \sum_{j \in D} v_j^* \leq 4 \cdot \text{OPT}$.

This analysis is a little unsatisfactory in the sense that we bound $\sum_{i \in F} f_i y_i^*$ by OPT, whereas we know that we have the stronger bound $\sum_{i \in F} f_i y_i^* + \sum_{i \in F, j \in D} c_{ij} x_{ij}^* \leq$ OPT. In this section, we show that by using randomized rounding we can modify the algorithm of Section 4.5 slightly and improve the analysis to a 3-approximation algorithm.

The basic idea is that once we have selected a client j in the algorithm, instead of opening the cheapest facility in $N(j)$, we use randomized rounding to choose the facility, and open facility $i \in N(j)$ with probability x_{ij}^* (note that $\sum_{i \in N(j)} x_{ij}^* = 1$). This improves the analysis since in the previous version of the algorithm we had to make worst-case assumptions about how far away the cheapest facility would be from the clients assigned to it. In this algorithm we can amortize the costs over all possible choices of facilities in $N(j)$.

In order to get our analysis to work, we modify the choice of client selected in each iteration as well. We define $C_j^* = \sum_{i \in F} c_{ij} x_{ij}^*$; that is, the assignment cost incurred by client j in the LP solution (x^*, y^*). We now choose the unassigned client that minimizes $v_j^* + C_j^*$ over all unassigned clients in each iteration. Our new algorithm is given in Algorithm 5.1. Note that the only changes from the previous algorithm of Section 4.5 (Algorithm 4.2) are in the third from the last and second from the last lines.

We can now analyze this new algorithm.

Theorem 5.19. *Algorithm 5.1 is a randomized 3-approximation algorithm for the uncapacitated facility location problem.*

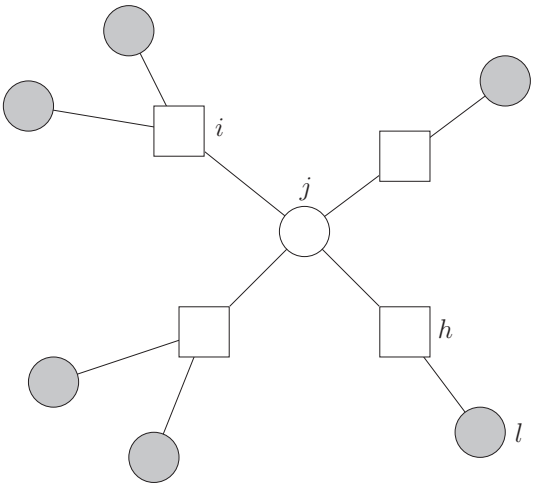

Figure 5.5. Illustration of proof of Theorem 5.19.

Proof. In an iteration k, the expected cost of the facility opened is

$$\sum_{i \in N(j_k)} f_i x^*_{ij_k} \leq \sum_{i \in N(j_k)} f_i y^*_i,$$

using the LP constraint $x^*_{ij_k} \leq y^*_i$. As we argued in Section 4.5, the neighborhoods $N(j_k)$ form a partition of a subset of the facilities so that the overall expected cost of facilities opened is at most

$$\sum_k \sum_{i \in N(j_k)} f_i y^*_i \leq \sum_{i \in F} f_i y^*_i.$$

We now fix an iteration k and let j denote the client j_k selected and let i denote the facility i_k opened. The expected cost of assigning j to i is

$$\sum_{i \in N(j)} c_{ij} x^*_{ij} = C^*_j.$$

As can be seen from Figure 5.5, the expected cost of assigning an unassigned client $l \in N^2(j)$ to i, where the client l neighbors facility h, which neighbors client j, is at most

$$c_{hl} + c_{hj} + \sum_{i \in N(j)} c_{ij} x^*_{ij} = c_{hl} + c_{hj} + C^*_j.$$

By Lemma 4.11, $c_{hl} \leq v^*_l$ and $c_{hj} \leq v^*_j$, so that this cost is at most $v^*_l + v^*_j + C^*_j$. Then since we chose j to minimize $v^*_j + C^*_j$ among all unassigned clients, we know that $v^*_j + C^*_j \leq v^*_l + C^*_l$. Hence, the expected cost of assigning l to i is at most

$$v^*_l + v^*_j + C^*_j \leq 2v^*_l + C^*_l.$$

Thus, we have that our total expected cost is no more than

$$\sum_{i \in F} f_i y_i^* + \sum_{j \in D} (2v_j^* + C_j^*) = \sum_{i \in F} f_i y_i^* + \sum_{i \in F, j \in D} c_{ij} x_{ij}^* + 2 \sum_{j \in D} v_j^* \leq 3\,\mathrm{OPT}.$$

□

Note that we were able to reduce the performance guarantee from 4 to 3 because the random choice of facility allows us to include the assignment cost C_j^* in the analysis; instead of bounding only the facility cost by OPT, we can bound both the facility cost and part of the assignment cost by OPT.

One can imagine a different type of randomized rounding algorithm: suppose we obtain an optimal LP solution (x^*, y^*) and open each facility $i \in F$ with probability y_i^*. Given the open facilities, we then assign each client to the closest open facility. This algorithm has the nice feature that the expected facility cost is $\sum_{i \in F} f_i y_i^*$. However, this simple algorithm clearly has the difficulty that with nonzero probability, the algorithm opens no facilities at all, and hence the expected assignment cost is unbounded. We consider a modified version of this algorithm later in the book, in Section 12.1.

5.9 Scheduling a Single Machine with Release Dates

In this section, we return to the problem considered in Section 4.2 of scheduling a single machine with release dates so as to minimize the weighted sum of completion times. Recall that we are given as input n jobs, each of which has a processing time $p_j > 0$, weight $w_j \geq 0$, and release date $r_j \geq 0$. The values p_j, r_j, and w_j are all nonnegative integers. We must construct a schedule for these jobs on a single machine such that at most one job is processed at any point in time, no job is processed before its release date, and once a job begins to be processed, it must be processed *nonpreemptively*; that is, it must be processed completely before any other job can be scheduled. If C_j denotes the time at which job j is finished processing, then the goal is to find the schedule that minimizes $\sum_{j=1}^{n} w_j C_j$.

In Section 4.2, we gave a linear programming relaxation of the problem. In order to apply randomized rounding, we will use a different integer programming formulation of this problem. In fact, we will not use an integer programming formulation of the problem, but an integer programming *relaxation*. Solutions in which jobs are scheduled preemptively are feasible; however, the contribution of job j to the objective function is less than $w_j C_j$ unless job j is scheduled nonpreemptively. Thus, the integer program is a relaxation since for any solution corresponding to a nonpreemptive schedule, the objective function value is equal to the sum of weighted completion times of the schedule.

Furthermore, although this relaxation is an integer program and has a number of constraints and variables exponential in the size of the problem instance, we will be able to find a solution to it in polynomial time.

We now give the integer programming relaxation. Let T equal $\max_j r_j + \sum_{j=1}^{n} p_j$, which is the latest possible time any job can be processed in any schedule that processes

a job nonpreemptively whenever it can. We introduce variables y_{jt} for $j = 1, \ldots, n$, $t = 1, \ldots, T$, where

$$y_{jt} = \begin{cases} 1 & \text{if job } j \text{ is processed in time } [t-1, t), \\ 0 & \text{otherwise.} \end{cases}$$

We derive a series of constraints for the integer program to capture the constraints of the scheduling problem. Since at most one job can be processed at any point in time, for each time $t = 1, \ldots, T$ we impose the constraint

$$\sum_{j=1}^{n} y_{jt} \leq 1.$$

Since each job j must be processed for p_j units of time, for each job $j = 1, \ldots, n$ we impose the constraint

$$\sum_{t=1}^{T} y_{jt} = p_j.$$

Since no job j can be processed before its release date, we set

$$y_{jt} = 0$$

for each job $j = 1, \ldots, n$ and each time $t = 1, \ldots, r_j$. For $t = r_j + 1, \ldots, T$, obviously we want

$$y_{jt} \in \{0, 1\}.$$

Note that this integer program has size that is exponential in the size of the scheduling instance because T is exponential in the number of bits used to encode r_j and p_j.

Finally, we need to express the completion time of job j in terms of the variables y_{jt}, $j = 1, \ldots, n$. Given a nonpreemptive schedule, suppose that job j completes at time D. If we set the variables y_{jt} to indicate the times at which jobs are processed in the schedule, and so $y_{jt} = 1$ for $t = D - p_j + 1, \ldots, D$, whereas $y_{jt} = 0$ otherwise. Observe that if we take the average of the midpoints of each unit of time at which job j is processed ($t - \frac{1}{2}$ for $t = D - p_j + 1, \ldots, D$), we get the midpoint of the processing time of the job, namely, $D - \frac{p_j}{2}$. Thus,

$$\frac{1}{p_j} \sum_{t=D-p_j+1}^{D} \left(t - \frac{1}{2}\right) = D - \frac{p_j}{2}.$$

Given the settings of the variables y_{jt}, we can rewrite this as

$$\frac{1}{p_j} \sum_{t=1}^{T} y_{jt} \left(t - \frac{1}{2}\right) = D - \frac{p_j}{2}.$$

We wish to have variable C_j represent the completion time of job j. Rearranging terms, then, we set the variable C_j as follows:

$$C_j = \frac{1}{p_j} \sum_{t=1}^{T} y_{jt} \left(t - \frac{1}{2}\right) + \frac{p_j}{2}.$$

This variable C_j underestimates the completion time of job j when all of the variables y_{jt} that are set to 1 are not consecutive in time. To see this, first start with the case above in which $y_{jt} = 1$ for $t = D - p_j + 1, \ldots, D$ for a completion time D. By the arguments above, $C_j = D$. If we then modify the variables y_{jt} by successively setting $y_{jt} = 0$ for some $t \in [D - p_j + 1, D - 1]$ and $y_{jt} = 1$ for some $t \leq D - p_j$, it is clear that the variable C_j only decreases.

The overall *integer* programming relaxation of the problem we will use is

$$\text{minimize} \quad \sum_{j=1}^{n} w_j C_j \tag{5.3}$$

$$\text{subject to} \quad \sum_{j=1}^{n} y_{jt} \leq 1, \qquad t = 1, \ldots, T, \tag{5.4}$$

$$\sum_{t=1}^{T} y_{jt} = p_j, \qquad j = 1, \ldots, n, \tag{5.5}$$

$$y_{jt} = 0, \qquad j = 1, \ldots, n; t = 1, \ldots, r_j,$$

$$y_{jt} \in \{0, 1\}, \qquad j = 1, \ldots, n; t = 1, \ldots, T,$$

$$C_j = \frac{1}{p_j} \sum_{t=1}^{T} y_{jt} \left(t - \frac{1}{2} \right) + \frac{p_j}{2}, \quad j = 1, \ldots, n. \tag{5.6}$$

Even though we restrict the variables y_{jt} to take on integer values, the corresponding linear programming relaxation is well known to have optimal solutions for which these variables have integer values (for a simpler case of this phenomenon, see Exercise 4.6).

We can now consider a randomised rounding algorithm for this problem. Let (y^*, C^*) be an optimal solution to the integer programming relaxation. For each job j, let X_j be a random variable that is $t - \frac{1}{2}$ with probability y^*_{jt}/p_j; observe that by (5.5), $\sum_{t=1}^{T} \frac{y^*_{jt}}{p_j} = 1$ so that the y^*_{jt}/p_j give a probability distribution on time t for each job j. We defer for the moment a discussion on how to make this run in randomized polynomial time, since T is exponential in the input size and we need to solve the integer program in order to perform the randomized rounding. As in the algorithms of Sections 4.1 and 4.2 we now schedule the jobs as early as possible in the same relative order as the value of the X_j. Without loss of generality, suppose that $X_1 \leq X_2 \leq \cdots \leq X_n$. Then we schedule job 1 as early as possible (that is, not before r_1), then job 2, and in general we schedule job j to start at the maximum of the completion time of job $j - 1$ and r_j. Let $\hat{C}_j$ be a random variable denoting the completion time of job j in this schedule.

We begin the analysis of this algorithm by considering the expected value of $\hat{C}_j$ given a fixed value of X_j.

Lemma 5.20. $E[\hat{C}_j | X_j = x] \leq p_j + 2x$.

Proof. As we argued in the proof of Lemma 4.2, there cannot be any idle time between $\max_{k=1,\ldots,j} r_k$ and $\hat{C}_j$, and therefore it must be the case that $\hat{C}_j \leq \max_{k=1,\ldots,j} r_k + \sum_{k=1}^{j} p_j$. Because the ordering of the jobs results from randomized rounding, we

let R be a random variable such that $R = \max_{k=1,\ldots,j} r_k$, and let random variable $P = \sum_{k=1}^{j-1} p_j$, so that the bound on $\hat{C}_j$ becomes $\hat{C}_j \leq R + P + p_j$.

First, we bound the value of R given that $X_j = x$. Note that since $y^*_{kt} = 0$ for $t \leq r_k$, it must be the case that $X_k \geq r_k + \frac{1}{2}$ for any job k. Thus,

$$R \leq \max_{k:X_k \leq X_j} r_k \leq \max_{k:X_k \leq X_j} X_k - \frac{1}{2} \leq X_j - \frac{1}{2} = x - \frac{1}{2}.$$

Now we bound the expected value of P given that $X_j = x$. We can bound it as follows:

$$\begin{aligned} E[P|X_j = x] &= \sum_{k:k\neq j} p_k \Pr[\text{job } k \text{ is processed before } j | X_j = x] \\ &= \sum_{k:k\neq j} p_k \Pr[X_k \leq X_j | X_j = x] \\ &= \sum_{k:k\neq j} p_k \sum_{t=1}^{x+\frac{1}{2}} \Pr\left[X_k = t - \frac{1}{2}\right]. \end{aligned}$$

Since we set $X_k = t - \frac{1}{2}$ with probability y^*_{kt}/p_k, then

$$E[P|X_j = x] = \sum_{k:k\neq j} p_k \left(\sum_{t=1}^{x+\frac{1}{2}} \frac{y^*_{kt}}{p_k} \right) = \sum_{k:k\neq j} \sum_{t=1}^{x+\frac{1}{2}} y^*_{kt} = \sum_{t=1}^{x+\frac{1}{2}} \sum_{k:k\neq j} y^*_{kt}.$$

Constraint (5.4) of the integer programming relaxation imposes that $\sum_{k:k\neq j} y_{kt} \leq 1$ for all times t, so then

$$E[P|X_j = x] = \sum_{t=1}^{x+\frac{1}{2}} \sum_{k:k\neq j} y^*_{kt} \leq x + \frac{1}{2}.$$

Therefore,

$$\begin{aligned} E[\hat{C}_j|X_j = x] &\leq p_j + E[R|X_j = x] + E[P|X_j = x] \\ &\leq p_j + \left(x - \frac{1}{2}\right) + \left(x + \frac{1}{2}\right) = p_j + 2x. \end{aligned}$$

□

Given the lemma above, we can prove the following theorem.

Theorem 5.21. *The randomized rounding algorithm is a randomized 2-approximation algorithm for the single-machine scheduling problem with release dates minimizing the sum of weighted completion times.*

Proof. Using the lemma above, we have that

$$\begin{aligned} E[\hat{C}_j] &= \sum_{t=1}^{T} E\left[\hat{C}_j \,\middle|\, X_j = t - \frac{1}{2}\right] \Pr\left[X_j = t - \frac{1}{2}\right] \\ &\leq p_j + 2\sum_{t=1}^{T}\left(t - \frac{1}{2}\right)\Pr\left[X_j = t - \frac{1}{2}\right] \\ &= p_j + 2\sum_{t=1}^{T}\left(t - \frac{1}{2}\right)\frac{y_{jt}^*}{p_j} \\ &= 2\left[\frac{p_j}{2} + \frac{1}{p_j}\sum_{t=0}^{T-1}\left(t - \frac{1}{2}\right)y_{jt}^*\right] \\ &= 2C_j^*, \end{aligned} \tag{5.7}$$

where equation (5.7) follows from the definition of C_j^* in the integer programming relaxation (equation (5.6)). Thus, we have that

$$E\left[\sum_{j=1}^{n} w_j \hat{C}_j\right] = \sum_{j=1}^{n} w_j E[\hat{C}_j] \leq 2\sum_{j=1}^{n} w_j C_j^* \leq 2\,\mathrm{OPT},$$

since $\sum_{j=1}^{n} w_j C_j^*$ is the objective function of the integer programming relaxation and thus a lower bound on OPT. □

Unlike the previous randomized algorithms of the section, we do not know directly how to derandomize this algorithm, although a deterministic 2-approximation algorithm for this problem does exist.

We now show that the integer programming relaxation can be solved in polynomial time, and that the randomized rounding algorithm can be made to run in polynomial time. First, sort the jobs in order of non-increasing ratio of weight to processing time; we assume jobs are then indexed in this order so that $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \cdots \geq \frac{w_n}{p_n}$. Now we use the following rule to create a (possibly preemptive) schedule: we always schedule the job of minimum index that is available but not yet completely processed. More formally, as t varies from 1 to T, let j be the smallest index such that $r_j \leq t - 1$ and $\sum_{z=1}^{t-1} y_{jz}^* < p_j$, if such a job j exists. Then set $y_{jt}^* = 1$ for job j and $y_{kt}^* = 0$ for all jobs $k \neq j$. If no such j exists, we set $y_{jt}^* = 0$ for all jobs j. Since in creating this schedule there are only n points in time corresponding to release dates r_j and n points in time at which a job has finished being processed, there are at most $2n$ points in time at which the index attaining the minimum might change. Thus, we can actually give the schedule as a sequence of at most $2n$ intervals of time, specifying which job (if any) is scheduled for each interval. It is not hard to see that we can compute this set of intervals in polynomial time without explicitly enumerating each time t. Furthermore, from the discussion above in which we explained how to express the variable C_j in terms of the y_{jt}, we know that if $y_{jt} = 1$ for $t = a + 1$ to b, then $\sum_{t=a+1}^{b} y_{jt}\left(t - \frac{1}{2}\right) = (b - a)(b - \frac{1}{2}(b - a))$, so that we can compute the values of the variables C_j^* from these intervals.

The randomized rounding algorithm can be made to run in polynomial time because it is equivalent to the following algorithm: for each job j, choose a value $\alpha_j \in [0, 1]$ independently and uniformly. Let X_j be the *α_j-point* of job j, that is, the time when $\alpha_j p_j$ units of job j have been processed in the preemptive schedule. Observe that it is easy to compute this point in time from the intervals describing the preemptive schedule. Then schedule the jobs according to the ordering of the X_j as in the randomized rounding algorithm. To see why this is equivalent to the original algorithm, consider the probability that $X_j \in [t-1, t)$. This is simply the probability that $\alpha_j p_j$ units of job j have finished processing in this interval, which is the probability that

$$\sum_{s=1}^{t-1} y_{js}^* \leq \alpha_j p_j < \sum_{s=1}^{t} y_{js}^*.$$

This, then, is the probability that $\alpha_j \in [\frac{1}{p_j}\sum_{s=1}^{t-1} y_{js}^*, \frac{1}{p_j}\sum_{s=1}^{t} y_{js}^*)$; since α_j is chosen uniformly, this probability is y_{jt}^*/p_j. So the probability that $X_j \in [t-1, t)$ in the α_j-point algorithm is the same as in the original algorithm, and the proof of the performance guarantee goes through with some small modifications.

Interestingly, one can also prove that if a single value of α is chosen uniformly from $[0, 1]$, and X_j is the α-point of job j, then scheduling jobs according to the ordering of the X_j is also a 2-approximation algorithm. Proving this fact is beyond the scope of this section. However, the algorithm in which a single value of α is chosen is easy to derandomize because it is possible to show that at most n different schedules can result from all possible choices of $\alpha \in [0, 1]$. Then to derive a deterministic 2-approximation algorithm, we need only enumerate the n different schedules, and choose the one that minimizes the weighted sum of the completion times.

It remains to show that the constructed solution (y^*, C^*) is in fact optimal. This is left to the reader to complete by a straightforward interchange argument in Exercise 5.11.

Lemma 5.22. *The solution (y^*, C^*) given above to the integer program is an optimal solution.*

5.10 Chernoff Bounds

This section introduces some theorems that are extremely useful for analyzing randomized rounding algorithms. In essence, the theorems say that it is very likely that the sum of n independent 0-1 random variables is not far away from the expected value of the sum. In the subsequent two sections, we will illustrate the usefulness of these bounds.

We begin by stating the main theorems.

Theorem 5.23. *Let $X_1, \ldots, X_n$ be n independent 0-1 random variables, not necessarily identically distributed. Then for $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$, $L \leq \mu \leq U$, and $\delta > 0$,*

$$\Pr[X \geq (1+\delta)U] < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{U},$$

and

$$\Pr[X \le (1-\delta)L] < \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^L.$$

The second theorem generalizes the first by replacing 0-1 random variables with 0-a_i random variables, where $0 < a_i \le 1$.

Theorem 5.24. *Let $X_1, \ldots, X_n$ be n independent random variables, not necessarily identically distributed, such that each X_i takes either the value 0 or the value a_i for some $0 < a_i \le 1$. Then for $X = \sum_{i=1}^n X_i$ and $\mu = E[X]$, $L \le \mu \le U$, and $\delta > 0$,*

$$\Pr[X \ge (1+\delta)U] < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^U,$$

and

$$\Pr[X \le (1-\delta)L] < \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^L.$$

These theorems are generalizations of results due to Chernoff and are sometimes called *Chernoff bounds*, since they bound the probability that the sum of variables is far away from its mean.

To prove the bounds, we will need the following commonly used inequality known as *Markov's inequality*.

Lemma 5.25 (Markov's inequality). *If X is a random variable taking on nonnegative values, then $\Pr[X \ge a] \le E[X]/a$ for $a > 0$.*

Proof. Since X takes on nonnegative values, $E[X] \ge a \Pr[X \ge a]$, and the inequality follows. □

Now we can prove Theorem 5.24.

Proof of Theorem 5.24. We prove only the first bound in the theorem; the proof of the other bound is analogous. Note that if $E[X] = 0$, then $X = 0$ and the bound holds trivially, so we can assume $E[X] > 0$ and $E[X_i] > 0$ for some i. We ignore all i such that $E[X_i] = 0$ since $X_i = 0$ for such i. Let $p_i = \Pr[X_i = a_i]$. Since $E[X_i] > 0$, $p_i > 0$. Then $\mu = E[X] = \sum_{i=1}^n p_i a_i \le U$. For any $t > 0$,

$$\Pr[X \ge (1+\delta)U] = \Pr[e^{tX} \ge e^{t(1+\delta)U}].$$

By Markov's inequality,

$$\Pr[e^{tX} \ge e^{t(1+\delta)U}] \le \frac{E[e^{tX}]}{e^{t(1+\delta)U}}. \tag{5.8}$$

Now

$$E[e^{tX}] = E\left[e^{t\sum_{i=1}^n X_i}\right] = E\left[\prod_{i=1}^n e^{tX_i}\right] = \prod_{i=1}^n E[e^{tX_i}], \tag{5.9}$$

where the equality follows by the independence of the X_i. Then for each i,

$$E[e^{tX_i}] = (1-p_i) + p_i e^{ta_i} = 1 + p_i(e^{ta_i} - 1).$$

We will show that $e^{ta_i} - 1 \leq a_i(e^t - 1)$ for $t > 0$, so that $E[e^{tX_i}] \leq 1 + p_i a_i(e^t - 1)$. Using that $1 + x < e^x$ for $x > 0$, and that $t, a_i, p_i > 0$, we obtain

$$E[e^{tX_i}] < e^{p_i a_i(e^t - 1)}.$$

Plugging these back into equation (5.9), we have

$$E[e^{tX}] < \prod_{i=1}^{n} e^{p_i a_i(e^t - 1)} = e^{\sum_{i=1}^{n} p_i a_i(e^t - 1)} \leq e^{U(e^t - 1)}.$$

Then putting this back into inequality (5.8) and setting $t = \ln(1 + \delta) > 0$, we see that

$$\begin{aligned} \Pr[X \geq (1 + \delta)U] &\leq \frac{E[e^{tX}]}{e^{t(1+\delta)U}} \\ &< \frac{e^{U(e^t - 1)}}{e^{t(1+\delta)U}} \\ &= \left(\frac{e^{\delta}}{(1 + \delta)^{(1+\delta)}}\right)^{U}, \end{aligned}$$

as desired.

Finally, to see that $e^{a_i t} - 1 \leq a_i(e^t - 1)$ for $t > 0$, let $f(t) = a_i(e^t - 1) - e^{a_i t} - 1$. Then $f'(t) = a_i e^t - a_i e^{a_i t} \geq 0$ for any $t \geq 0$ given that $0 < a_i \leq 1$; thus, $f(t)$ is non-decreasing for $t \geq 0$. Since $f(0) = 0$ and the function f is non-decreasing, the inequality holds. □

The right-hand sides of the inequalities in Theorems 5.23 and 5.24 are a bit complicated, and so it will be useful to consider variants of the results in which the right-hand side is simpler, at the cost of restricting the results somewhat.

Lemma 5.26. *For $0 \leq \delta \leq 1$, we have that*

$$\left(\frac{e^{\delta}}{(1 + \delta)^{(1+\delta)}}\right)^{U} \leq e^{-U\delta^2/3},$$

and for $0 \leq \delta < 1$, we have that

$$\left(\frac{e^{-\delta}}{(1 - \delta)^{(1-\delta)}}\right)^{L} \leq e^{-L\delta^2/2}.$$

Proof. For the first inequality, we take the logarithm of both sides. We would like to show that

$$U(\delta - (1 + \delta)\ln(1 + \delta)) \leq -U\delta^2/3.$$

We observe that the inequality holds for $\delta = 0$; if we can show that the derivative of the left-hand side is no more than that of the right-hand side for $0 \leq \delta \leq 1$, the inequality will hold for $0 \leq \delta \leq 1$. Taking derivatives of both sides, we need to show that

$$-U\ln(1 + \delta) \leq -2U\delta/3.$$

Letting $f(\delta) = -U\ln(1 + \delta) + 2U\delta/3$, we need to show that $f(\delta) \leq 0$ on $[0, 1]$. Note that $f(0) = 0$ and $f(1) \leq 0$ since $-\ln 2 \approx -0.693 < -2/3$. As long as the function $f(\delta)$ is convex on $[0, 1]$ (that is, $f''(\delta) \geq 0$), we may conclude that $f(\delta) \leq 0$ on $[0, 1]$,

in the convex analog of Fact 5.9. We observe that $f'(\delta) = -U/(1+\delta) + 2U/3$ and $f''(\delta) = U/(1+\delta)^2$, so that $f''(\delta) \geq 0$ for $\delta \in [0, 1]$, and the inequality is shown.

We turn to the second inequality for the case $0 \leq \delta < 1$. Taking the logarithm of both sides, we would like to show that

$$L(-\delta - (1-\delta)\ln(1-\delta)) \leq -L\delta^2/2.$$

The inequality holds for $\delta = 0$, and will hold for $0 < \delta < 1$ if the derivative of the left-hand side is no more than the derivative of the right-hand side for $0 \leq \delta < 1$. Taking derivatives of both sides, we would like to show that

$$L\ln(1-\delta) \leq -L\delta$$

for $0 \leq \delta < 1$. Again the inequality holds for $\delta = 0$ and will hold for $0 \leq \delta < 1$ if the derivative of the left-hand side is no more than the derivative of the right-hand side for $0 \leq \delta < 1$. Taking derivatives of both sides again, we obtain

$$-L/(1-\delta) \leq -L,$$

which holds for $0 \leq \delta < 1$. □

It will sometimes be useful to provide a bound on the probability that $X \leq (1-\delta)L$ in the case $\delta = 1$. Notice that since the variables X_i are either 0-1 or 0-a_i this is asking for a bound on the probability that $X = 0$. We can give a bound as follows.

Lemma 5.27. *Let $X_1, \ldots, X_n$ be n independent random variables, not necessarily identically distributed, such that each X_i takes either the value 0 or the value a_i for some $0 < a_i \leq 1$. Then for $X = \sum_{i=1}^n X_i$ and $\mu = E[X]$, $L \leq \mu$,*

$$\Pr[X = 0] < e^{-L}.$$

Proof. We assume $\mu = E[X] > 0$ since otherwise $X = 0$ and $L \leq \mu = 0$ and the bound holds trivially. Let $p_i = \Pr[X_i = a_i]$. Then $\mu = \sum_{i=1}^n a_i p_i$ and

$$\Pr[X = 0] = \prod_{i=1}^n (1 - p_i).$$

Applying the arithmetic-geometric mean inequality from Fact 5.8, we get that

$$\prod_{i=1}^n (1-p_i) \leq \left[\frac{1}{n}\sum_{i=1}^n (1-p_i)\right]^n = \left[1 - \frac{1}{n}\sum_{i=1}^n p_i\right]^n.$$

Since each $a_i \leq 1$, we then obtain

$$\left[1 - \frac{1}{n}\sum_{i=1}^n p_i\right]^n \leq \left[1 - \frac{1}{n}\sum_{i=1}^n a_i p_i\right]^n = \left[1 - \frac{1}{n}\mu\right]^n.$$

Then using the fact that $1 - x < e^{-x}$ for $x > 0$, we get

$$\left[1 - \frac{1}{n}\mu\right]^n < e^{-\mu} \leq e^{-L}.$$

□

As a corollary, we can then extend the bound of Lemma 5.26 to the case $\delta = 1$. In fact, since the probability that $X < (1-\delta)L$ for $\delta > 1$ and $L \geq 0$ is 0, we can extend the bound to any positive δ.

Corollary 5.28. *Let $X_1, \ldots, X_n$ be n independent random variables, not necessarily identically distributed, such that each X_i takes either the value 0 or the value a_i for some $a_i \leq 1$. Then for $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$, $0 \leq L \leq \mu$, and $\delta > 0$,*

$$\Pr[X \leq (1-\delta)L] < e^{-L\delta^2/2}.$$

5.11 Integer Multicommodity Flows

To see how Chernoff bounds can be used in the context of randomized rounding, we will apply them to the *minimum-capacity multicommodity flow problem*. In this problem, we are given as input an undirected graph $G = (V, E)$ and k pairs of vertices $s_i, t_i \in V$, $i = 1, \ldots, k$. The goal is to find, for each $i = 1, \ldots, k$, a single simple path from s_i to t_i so as to minimize the maximum number of paths containing the same edge. This problem arises in routing wires on chips. In this problem, k wires need to be routed, each wire from some point s_i on the chip to another point t_i on the chip. Wires must be routed through channels on the chip, which correspond to edges in a graph. The goal is to route the wires so as minimize the channel capacity needed, that is, the number of wires routed through the same channel. The problem as stated here is a special case of a more interesting problem, since often the wires must join up three or more points on a chip.

We give an integer programming formulation of the problem. Let $\mathcal{P}_i$ be the set of all possible simple paths P in G from s_i to t_i, where P is the set of edges in the path. We create a 0-1 variable x_P for each path $P \in \mathcal{P}_i$ to indicate when path P from s_i to t_i is used. Then the total number of paths using an edge $e \in E$ is simply $\sum_{P:e\in P} x_P$. We create another decision variable W to denote the maximum number of paths using an edge, so that our objective function is to minimize W. We have the constraint that

$$\sum_{P:e\in P} x_P \leq W$$

for each edge $e \in E$. Finally, we need to choose some path $P \in \mathcal{P}_i$ for every s_i-t_i pair, so that

$$\sum_{P\in\mathcal{P}_i} x_P = 1$$

for each $i = 1, \ldots, k$. This gives us the following integer programming formulation of the minimum-capacity multicommodity flow problem:

$$\text{minimize} \quad W \tag{5.10}$$

$$\begin{aligned} \text{subject to} \quad & \sum_{P\in\mathcal{P}_i} x_P = 1, && i = 1, \ldots, k, \\ & \sum_{P:e\in P} x_P \leq W, && e \in E, \\ & x_P \in \{0,1\}, && \forall P \in \mathcal{P}_i, i = 1, \ldots, k. \end{aligned} \tag{5.11}$$

The integer program can be relaxed to a linear program by replacing the constraints $x_P \in \{0, 1\}$ with $x_P \geq 0$. We claim for now that this linear program can be solved in polynomial time and that at most a polynomial number of variables x_P of an optimal solution can be nonzero.

We now apply randomized rounding to obtain a solution to the problem. For $i = 1, \ldots, k$, we choose exactly one path $P \in \mathcal{P}_i$ according to the probability distribution x_P^* on paths $P \in \mathcal{P}_i$, where x^* is an optimal solution of value W^*.

Assuming W^* is large enough, we can show that the total number of paths going through any edge is close to W^* by using the Chernoff bound from Theorem 5.23. Let n be the number of vertices in the graph. Recall that in Section 1.7 we said that a probabilistic event happens with high probability if the probability that it does not occur is at most n^{-c} for some $c \geq 1$.

Theorem 5.29. *If $W^* \geq c \ln n$ for some constant c, then with high probability, the total number of paths using any edge is at most $W^* + \sqrt{cW^* \ln n}$.*

Proof. For each $e \in E$, define random variables X_e^i, where $X_e^i = 1$ if the chosen s_i-t_i path uses edge e, and $X_e^i = 0$ otherwise. Then the number of paths using edge e is $Y_e = \sum_{i=1}^{k} X_e^i$. We want to bound $\max_{e \in E} Y_e$, and show that this is close to the LP value W^*. Certainly

$$E[Y_e] = \sum_{i=1}^{k} \sum_{P \in \mathcal{P}_i : e \in P} x_P^* = \sum_{P : e \in P} x_P^* \leq W^*,$$

by constraint (5.11) from the LP. For a fixed edge e, the random variables X_e^i are independent, so we can apply the Chernoff bound of Theorem 5.23. Set $\delta = \sqrt{(c \ln n)/W^*}$. Since $W^* \geq c \ln n$ by assumption, it follows that $\delta \leq 1$. Then by Theorem 5.23 and Lemma 5.26 with $U = W^*$,

$$\Pr[Y_e \geq (1 + \delta)W^*] < e^{-W^* \delta^2/3} = e^{-(c \ln n)/3} = \frac{1}{n^{c/3}}.$$

Also $(1 + \delta)W^* = W^* + \sqrt{cW^* \ln n}$. Since there can be at most n^2 edges,

$$\begin{aligned} \Pr\left[\max_{e \in E} Y_e \geq (1 + \delta)W^*\right] &\leq \sum_{e \in E} \Pr[Y_e \geq (1 + \delta)W^*] \\ &\leq n^2 \cdot \frac{1}{n^{c/3}} = n^{2 - c/3}. \end{aligned}$$

For a constant $c \geq 12$, this ensures that the theorem statement fails to hold with probability at most $\frac{1}{n^2}$, and by increasing c we can make the probability as small as we like. □

Observe that since $W^* \geq c \ln n$, the theorem above guarantees that the randomized algorithm produces a solution of no more than $2W^* \leq 2\,\mathrm{OPT}$. However, the algorithm might produce a solution considerably closer to optimal if $W^* \gg c \ln n$. We also observe the following corollary.

Corollary 5.30. *If $W^* \geq 1$, then with high probability, the total number of paths using any edge is $O(\log n) \cdot W^*$.*

Proof. We repeat the proof above with $U = (c \ln n)W^*$ and $\delta = 1$. □

In fact, the statement of the corollary can be sharpened by replacing the $O(\log n)$ with $O(\log n / \log \log n)$ (see Exercise 5.13).

To solve the linear program in polynomial time, we show that it is equivalent to a polynomially sized linear program; we leave this as an exercise to the reader (Exercise 5.14, to be precise).

5.12 Random Sampling and Coloring Dense 3-Colorable Graphs

In this section we turn to another application of Chernoff bounds. We consider coloring a δ-dense 3-colorable graph. We say that a graph is *dense* if for some constant α the number of edges in the graph is at least $\alpha\binom{n}{2}$; in other words, some constant fraction of the edges that could exist in the graph do exist. A *δ-dense* graph is a special case of a dense graph. A graph is δ-dense if every node has at least δn neighbors for some constant δ; that is, every node has some constant fraction of the neighbors it could have. Finally, a graph is *k-colorable* if each of the nodes can be assigned exactly one of k colors in such a way that no edge has its two endpoints assigned the same color. In general, it is NP-complete to decide whether a graph is 3-colorable; in fact, the following is known.

Theorem 5.31. *It is NP-hard to decide if a graph can be colored with only three colors, or needs at least five colors.*

Theorem 5.32. *Assuming a variant of the unique games conjecture, for any constant $k > 3$, it is NP-hard to decide if a graph can be colored with only three colors, or needs at least k colors.*

In Sections 6.5 and 13.2 we will discuss approximation algorithms for coloring any 3-colorable graph. Here we will show that with high probability we can properly color any δ-dense 3-colorable graph in polynomial time. While this is not an approximation algorithm, it is a useful application of Chernoff bounds, which we will use again in Section 12.4.

In this case, we use the bounds to show that if we know the correct coloring for a small, randomly chosen sample of a δ-dense graph, we can give a polynomial-time algorithm that with high probability successfully colors the rest of the graph. This would seem to pose a problem, though, since we do not know the coloring for the sample. Nevertheless, if the sample is no larger than $O(\log n)$, we can enumerate all possible colorings of the sample in polynomial time, and run the algorithm above for each coloring. Since one of the colorings of the sample will be the correct one, for at least one of the possible colorings of the sample the algorithm will result in a correct coloring of the graph with high probability.

More specifically, given a δ-dense graph, we will select a random subset $S \subseteq V$ of $O((\ln n)/\delta)$ vertices by including each vertex in the subset with probability $(3c \ln n)/\delta n$ for some constant c. We will show first that the set size is no more than $(6c \ln n)/\delta$ with high probability, and then that with high probability, every vertex has at least one neighbor in S. Thus, given a correct coloring of S, we can use the information about the coloring of S to deduce the colors of the rest of the vertices. Since each vertex has

a neighbor in S, its color is restricted to be one of the two remaining colors, and this turns out to be enough of a restriction that we can infer the remaining coloring. Finally, although we do not know the correct coloring of S we can run this algorithm for each of the $3^{(6c \ln n)/\delta} = n^{O(c/\delta)}$ possible colorings of S. One of the colorings of S will be the correct coloring, and thus in at least one run of the algorithm we will be able to color the graph successfully.

Lemma 5.33. *With probability at most $n^{-c/\delta}$, the set S has size $|S| \geq (6c \ln n)/\delta$.*

Proof. We use the Chernoff bound (Theorem 5.23 and Lemma 5.26). Let X_i be a 0-1 random variable indicating whether vertex i is included in S. Then since each vertex is included with probability $3c \ln n/\delta n$, $\mu = E[\sum_{i=1}^{n} X_i] = (3c \ln n)/\delta$. Applying the lemma with $U = (3c \ln n)/\delta$, the probability that $|S| \geq 2U$ is at most $e^{-\mu/3} = n^{-c/\delta}$. □

Lemma 5.34. *The probability that a given vertex $v \notin S$ has no neighbor in S is at most n^{-3c}.*

Proof. Let X_i be a 0-1 random variable indicating whether the ith neighbor of v is in S or not. Then $\mu = E[\sum_i X_i] \geq 3c \ln n$, since v has at least δn neighbors. Then applying Lemma 5.27 with $L = 3c \ln n$, the probability that v has no neighbors in S is no more than $\Pr[\sum_i X_i = 0] \leq e^{-L} = n^{-3c}$. □

Corollary 5.35. *With probability at least $1 - 2n^{-(c-1)}$, $|S| \leq (6c \ln n)/\delta$ and every $v \notin S$ has at least one neighbor in S.*

Proof. This follows from Lemmas 5.33 and 5.34. The probability that both statements of the lemma are true is at least one minus the sum of the probabilities that either statement is false. The probability that every $v \notin S$ has no neighbor in S is at worst n times the probability that a given vertex $v \notin S$ has no neighbor in S. Since $\delta \leq 1$, the overall probability that both statements are true is at least

$$1 - n^{-c/\delta} - n \cdot n^{-3c} \geq 1 - 2n^{-(c-1)}.$$ □

Now we assume we have some coloring of the vertices in S, not necessarily one that is consistent with the correct coloring of the graph. We also assume that every vertex not in S has at least one neighbor in S. We further assume that the coloring of S is such that every edge with both endpoints in S has differently colored endpoints, since otherwise this is clearly not a correct coloring of the graph. Assume we color the graph with colors $\{0, 1, 2\}$. Given a vertex $v \notin S$, because it has some neighbor in S colored with some color $n(v) \in \{0, 1, 2\}$, we know that v cannot be colored with color $n(v)$. Possibly v has other neighbors in S with colors other than $n(v)$. Either this forces the color of v or there is no way we can successfully color v; in the latter case our current coloring of S must not have been correct, and we terminate. If the color of v is not determined, then we create a binary variable $x(v)$, which if true indicates that we color v with color $n(v) + 1 \pmod 3$, and if false indicates that we color v with color $n(v) - 1 \pmod 3$. Now every edge $(u, v) \in E$ for $u, v \notin S$ imposes the constraint $n(u) \neq n(v)$. To capture this, we create an instance of the maximum satisfiability problem such that all clauses are satisfiable if and only if the vertices not in S can be correctly colored. For each possible setting of the Boolean variables $x(u)$

and $x(v)$ that would cause $n(u) = n(v)$, we create a disjunction of $x(u)$ and $x(v)$ that is false if it implies $n(u) = n(v)$; for example, if $x(u) =$ true and $x(v) =$ false implies that $n(u) = n(v)$, then we create a clause $(\overline{x(u)} \vee x(v))$. Since G is 3-colorable, given a correct coloring of S, there exists a setting of the variables $x(v)$ that satisfies all the clauses. Since each clause has two variables, it is possible to determine in polynomial time whether the instance is satisfiable or not; we leave it as an exercise to the reader to prove this (Exercise 6.3). Obviously if we find a setting of the variables that satisfies all constraints, this implies a correct coloring of the entire graph, whereas if the constraints are not satisfiable, our current coloring of S must not have been correct.

In Section 12.4, we'll revisit the idea from this section of drawing a small random sample of a graph and using it to determine the overall solution for the maximum cut problem in dense graphs.

Exercises

5.1 In the *maximum k-cut problem*, we are given an undirected graph $G = (V, E)$, and nonnegative weights $w_{ij} \geq 0$ for all $(i, j) \in E$. The goal is to partition the vertex set V into k parts $V_1, \ldots, V_k$ so as to maximize the weight of all edges whose endpoints are in different parts (i.e., $\max_{(i,j)\in E: i\in V_a, j\in V_b, a\neq b} w_{ij}$).
Give a randomized $\frac{k-1}{k}$-approximation algorithm for the MAX k-CUT problem.

5.2 Consider the following greedy algorithm for the maximum cut problem. We suppose the vertices are numbered $1, \ldots, n$. In the first iteration, the algorithm places vertex 1 in U. In the kth iteration of the algorithm, we will place vertex k in either U or W. In order to decide which choice to make, we will look at all the edges F that have the vertex k as one endpoint and whose other endpoint is $1, \ldots, k-1$, so that $F = \{(j, k) \in E : 1 \leq j \leq k-1\}$. We choose to put vertex k in U or W depending on which of these two choices maximizes the number of edges of F being in the cut.

(a) Prove that this algorithm is a $1/2$-approximation algorithm for the maximum cut problem.
(b) Prove that this algorithm is equivalent to the derandomization of the maximum cut algorithm of Section 5.1 via the method of conditional expectations.

5.3 In the *maximum directed cut problem* (sometimes called MAX DICUT), we are given as input a directed graph $G = (V, A)$. Each directed arc $(i, j) \in A$ has nonnegative weight $w_{ij} \geq 0$. The goal is to partition V into two sets U and $W = V - U$ so as to maximize the total weight of the arcs going from U to W (that is, arcs (i, j) with $i \in U$ and $j \in W$). Give a randomized $\frac{1}{4}$-approximation algorithm for this problem.

5.4 Consider the nonlinear randomized rounding algorithm for MAX SAT as given in Section 5.6. Prove that using randomized rounding with the linear function $f(y_i) = \frac{1}{2}y_i + \frac{1}{4}$ also gives a $\frac{3}{4}$-approximation algorithm for MAX SAT.

5.5 Consider the nonlinear randomized rounding algorithm for MAX SAT as given in Section 5.6. Prove that using randomized rounding with the piecewise linear function

$$f(y_i) = \begin{cases} \frac{3}{4}y_i + \frac{1}{4} & \text{for } 0 \leq y_i \leq \frac{1}{3} \\ 1/2 & \text{for } \frac{1}{3} \leq y_i \leq \frac{2}{3} \\ \frac{3}{4}y_i & \text{for } \frac{2}{3} \leq y_i \leq 1 \end{cases}$$

also gives a $\frac{3}{4}$-approximation algorithm for MAX SAT.

5.6 Consider again the maximum directed cut problem from Exercise 5.3.

(a) Show that the following integer program models the maximum directed cut problem:

$$\begin{aligned}
\text{maximize} \quad & \sum_{(i,j)\in A} w_{ij} z_{ij} \\
\text{subject to} \quad & z_{ij} \leq x_i, && \forall (i,j) \in A, \\
& z_{ij} \leq 1 - x_j, && \forall (i,j) \in A, \\
& x_i \in \{0,1\}, && \forall i \in V, \\
& 0 \leq z_{ij} \leq 1, && \forall (i,j) \in A.
\end{aligned}$$

(b) Consider a randomized rounding algorithm for the maximum directed cut problem that solves a linear programming relaxation of the integer program and puts vertex $i \in U$ with probability $1/4 + x_i/2$. Show that this gives a randomized $1/2$-approximation algorithm for the maximum directed cut problem.

5.7 In this exercise, we consider how to derandomize the randomized rounding algorithm for the set cover problem given in Section 1.7. We would like to apply the method of conditional expectations, but we need to ensure that at the end of the process we obtain a valid set cover. Let X_j be a random variable indicating whether set S_j is included in the solution. Then if w_j is the weight of set S_j, let W be the weight of the set cover obtained by randomized rounding, so that $W = \sum_{j=1}^{m} w_j X_j$. Let Z be a random variable such that $Z = 1$ if randomized rounding does not produce a valid set cover, and $Z = 0$ if it does. Then consider applying the method of conditional expectations to the objective function $W + \lambda Z$ for some choice of $\lambda \geq 0$. Show that for the proper choice of λ, the method of conditional expectations applied to the randomized rounding algorithm yields an $O(\ln n)$-approximation algorithm for the set cover problem that always produces a set cover.

5.8 Consider a variation of the maximum satisfiability problem in which all variables occur positively in each clause, and there is an additional nonnegative weight $v_i \geq 0$ for each Boolean variable x_i. The goal is now to set the Boolean variables to maximize the total weight of the satisfied clauses plus the total weight of variables set to be false. Give an integer programming formulation for this problem, with 0-1 variables y_i to indicate whether x_i is set true. Show that a randomized rounding of the linear program in which variable x_i is set true with probability $1 - \lambda + \lambda y_i^*$ gives a $2(\sqrt{2} - 1)$-approximation algorithm for some appropriate setting of λ; note that $2(\sqrt{2} - 1) \approx 0.828$.

5.9 Recall the maximum coverage problem from Exercise 2.11; in it, we are given a set of elements E, and m subsets of elements $S_1, \ldots, S_m \subseteq E$ with a nonnegative weight $w_j \geq 0$ for each subset S_j. We would like to find a subset $S \subseteq E$ of size k that maximizes the total weight of the subsets covered by S, where S covers S_j if $S \cap S_j \neq \emptyset$.

(a) Show that the following nonlinear integer program models the maximum coverage problem:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j\in[m]} w_j \left(1 - \prod_{e\in S_j} (1 - x_e)\right) \\
\text{subject to} \quad & \sum_{e\in E} x_e = k, \\
& x_e \in \{0,1\}, \qquad \forall e \in E.
\end{aligned}$$

(b) Show that the following linear program is a relaxation of the maximum coverage problem:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j\in[m]} w_j z_j \\
\text{subject to} \quad & \sum_{e\in S_j} x_e \geq z_j, && \forall j \in [m], \\
& \sum_{e\in E} x_e = k, \\
& 0 \leq z_j \leq 1, && \forall j \in [m], \\
& 0 \leq x_e \leq 1, && \forall e \in E.
\end{aligned}$$

(c) Using the pipage rounding technique from Exercise 4.7, give an algorithm that deterministically rounds the optimal LP solution to an integer solution and has a performance guarantee of $1 - \frac{1}{e}$.

5.10 In the *uniform labeling problem*, we are given a graph $G = (V, E)$, costs $c_e \geq 0$ for all $e \in E$, and a set of labels L that can be assigned to the vertices of V. There is a nonnegative cost $c_v^i \geq 0$ for assigning label $i \in L$ to vertex $v \in V$, and an edge $e = (u, v)$ incurs cost c_e if u and v are assigned different labels. The goal of the problem is to assign each vertex in V a label so as to minimize the total cost.

We give an integer programming formulation of the problem. Let the variable $x_v^i \in \{0, 1\}$ be 1 if vertex v is assigned label $i \in L$, and 0 otherwise. Let the variable z_e^i be 1 if exactly one of the two endpoints of the edge e is assigned label i, and 0 otherwise. Then the integer programming formulation is as follows:

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{2}\sum_{e\in E} c_e \sum_{i\in L} z_e^i + \sum_{v\in V, i\in L} c_v^i x_v^i \\
\text{subject to} \quad & \sum_{i\in L} x_v^i = 1, && \forall v \in V, \\
& z_e^i \geq x_u^i - x_v^i, && \forall (u, v) \in E, \forall i \in L, \\
& z_e^i \geq x_v^i - x_u^i, && \forall (u, v) \in E, \forall i \in L, \\
& z_e^i \in \{0, 1\}, && \forall e \in E, \forall i \in L, \\
& x_v^i \in \{0, 1\}, && \forall v \in V, \forall i \in L.
\end{aligned}$$

(a) Prove that the integer programming formulation models the uniform labeling problem.

Consider now the following algorithm. First, the algorithm solves the linear programming relaxation of the integer program above. The algorithm then proceeds in phases. In each phase, it picks a label $i \in L$ uniformly at random, and a number $\alpha \in [0, 1]$ uniformly at random. For each vertex $v \in V$ that has not yet been assigned a label, we assign it label i if $\alpha \leq x_v^i$.

(b) Suppose that vertex $v \in V$ has not yet been assigned a label. Prove that the probability that v is assigned label $i \in L$ in the next phase is exactly $x_v^i/|L|$, and the probability that it is assigned a label in the next phase is exactly $1/|L|$. Further prove that the probability that v is assigned label i by the algorithm is exactly x_v^i.

(c) We say that an edge e is *separated by a phase* if both endpoints were not assigned labels prior to the phase, and exactly one of the endpoints is assigned

a label in this phase. Prove that the probability that an edge e is separated by a phase is $\frac{1}{|L|}\sum_{i\in L} z_e^i$.

(d) Prove that the probability that the endpoints of edge e receive different labels is at most $\sum_{i\in L} z_e^i$.

(e) Prove that the algorithm is a 2-approximation algorithm for the uniform labeling problem.

5.11 Prove Lemma 5.22, and show that the integer programming solution (y^*, C^*) described at the end of Section 5.9 must be optimal for the integer program (5.3).

5.12 Using randomized rounding and First Fit, give a randomized polynomial-time algorithm for the bin-packing problem that uses $\rho \cdot \mathrm{OPT}(I) + k$ bins for some $\rho < 2$ and some small constant k. One idea is to consider the linear program from Section 4.6.

5.13 Show that the $O(\log n)$ factor in Corollary 5.30 can be replaced with $O(\log n / \log\log n)$ by using Theorem 5.23.

5.14 Show that there is a linear programming relaxation for the integer multicommodity flow problem of Section 5.10 that is equivalent to the linear program (5.10) but has a number of variables and constraints that are bounded by a polynomial in the input size of the flow problem.

Chapter Notes

The textbooks of Mitzenmacher and Upfal [226] and Motwani and Raghavan [228] give more extensive treatments of randomized algorithms.

A 1967 paper of Erdős [99] on the maximum cut problem showed that sampling a solution uniformly at random as in Theorem 5.3 gives a solution whose expected value is at least half the sum of the edge weights. This is one of the first randomized approximation algorithms of which we are aware. This algorithm can also be viewed as a randomized version of a deterministic algorithm given by Sahni and Gonzalez [257] (the deterministic algorithm of Sahni and Gonzalez is given in Exercise 5.2).

Raghavan and Thompson [247] were the first to introduce the idea of the randomized rounding of a linear programming relaxation. The result for integer multicommodity flows in Section 5.11 is from their paper.

Random sampling and randomized rounding are most easily applied to unconstrained problems, such as the maximum satisfiability problem and the maximum cut problem, in which any solution is feasible. Even problems such as the prize-collecting Steiner tree problem and the uncapacitated facility location problem can be viewed as unconstrained problems: we need merely to select a set of vertices to span or facilities to open. Randomized approximation algorithms for constrained problems exist, but are much rarer.

The results for the maximum satisfiability problem in this chapter are due to a variety of authors. The simple randomized algorithm of Section 5.1 is given by Yannakakis [293] as a randomized variant of an earlier deterministic algorithm introduced by Johnson [179]. The "biased coins" algorithm of Section 5.3 is a similar randomization of an algorithm of Lieberherr and Specker [216]. The randomized rounding, "better of two," and nonlinear randomized rounding algorithms in Sections 5.4, 5.5, and 5.6, respectively, are due to Goemans and Williamson [137].

The derandomization of randomized algorithms is a major topic of study. The method of conditional expectations given in Section 5.2 is implicit in the work of Erdős and Selfridge [101], and has been developed by Spencer [271].

The randomized algorithm for the prize-collecting Steiner tree problem in Section 5.7 is an unpublished result of Goemans.

The algorithm of Section 5.8 for uncapacitated facility location is due to Chudak and Shmoys [77].

The scheduling algorithm of Section 5.9 is due to Schulz and Skutella [261]. The algorithm for solving the integer programming relaxation of this problem is due to Goemans [132], and the α-point algorithm that uses a single value of α is also due to Goemans [133].

Chernoff [71] gives the general ideas used in the proof of Chernoff bounds. Our proofs of these bounds follow those of Mitzenmacher and Upfal [226] and Motwani and Raghavan [228].

The randomized algorithm for 3-coloring a dense 3-colorable graph is due to Arora, Karger, and Karpinski [16]; a deterministic algorithm for the problem had earlier been given by Edwards [97]. Theorems 5.31 and 5.32 are due to Khanna, Linial, and Safra [190] (see also Guruswami and Khanna [152]) and Dinur, Mossel, and Regev [90], respectively.

Exercises 5.4 and 5.5 are due to Goemans and Williamson [137]. Exercise 5.6 is due to Trevisan [279, 280]. Exercise 5.7 is due to Norton [237]. Ageev and Sviridenko [1] gave the algorithm and analysis in Exercise 5.8, and Exercise 5.9 is also due to Ageev and Sviridenko [2, 3]. The algorithm for the uniform labeling problem in Exercise 5.10 is due to Kleinberg and Tardos [197]; the uniform labeling problem models a problem arising in image processing. Exercise 5.12 is an unpublished result of Williamson.