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 only holds 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 only know how to analyze 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 alternate 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, $j = 1, \dots, n$, 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 on.

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 3 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 an* $(\frac{7}{8}+\epsilon)$*-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,

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

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 only make the choice of x_1 deterministically, and 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 $E[W|x_1 \leftarrow \mathsf{true}] \geq E[W|x_1 \leftarrow \mathsf{false}]$, then we set x_1 true, otherwise we set it to false. Since by the definition of conditional expectations,

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

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

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

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

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

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

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

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 $p = \frac{1}{2}(\sqrt{5} - 1) \approx .618$. 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 zero 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

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

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:

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

If Z_{IP}^* is the optimum 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

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

If Z_{LP}^* is the optimum 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: *For any nonnegative $a_1, \ldots, a_k$,*

$$\sqrt[k]{\prod_{i=1}^{k} a_i} \leq \frac{1}{k} \sum_{i=1}^{k} a_i.$$

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

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

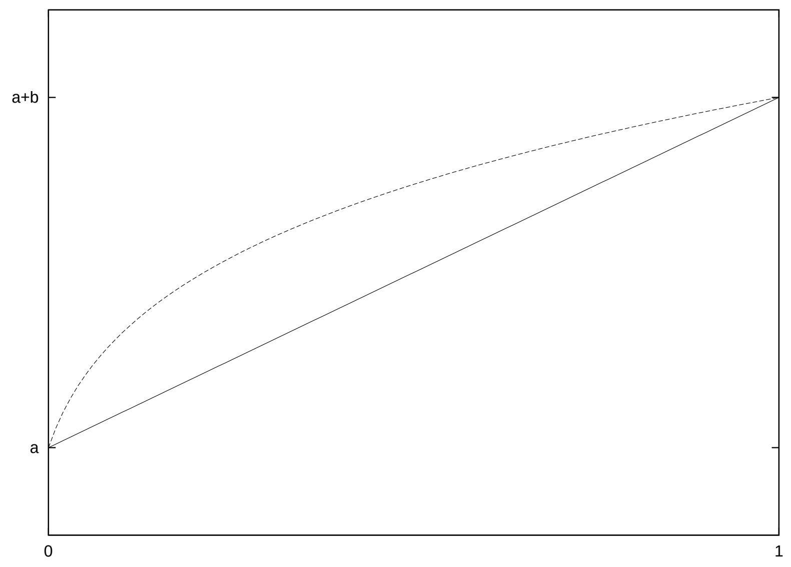

Figure 5.1: An illustration of Fact 5.9.

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 nonincreasing 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

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

□

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

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

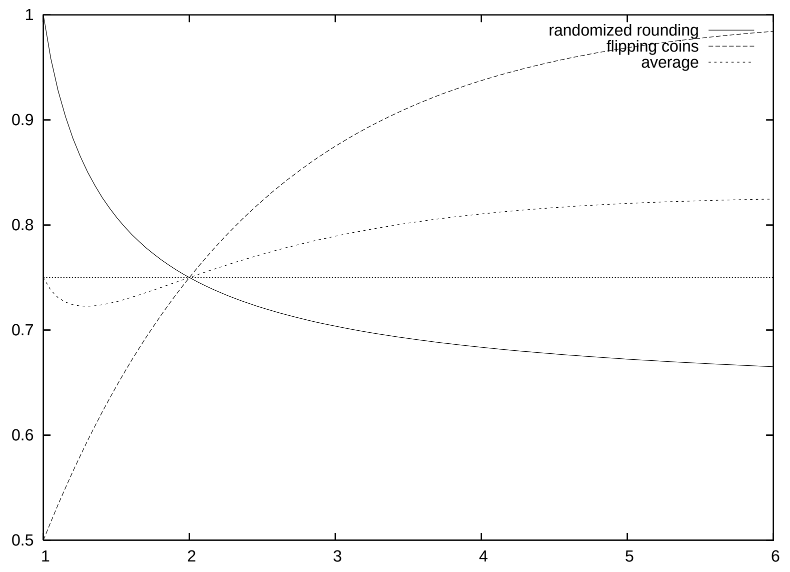

Figure 5.2: Illustration of the proof of Theorem 5.11. The "randomized rounding" line is the function $1-(1-\frac{1}{k})^k$. The "flipping coins" line is the function $1-2^{-k}$. The "average" line is the average of these two functions, which is at least $\frac{3}{4}$ for all integers $k \geq 1$.

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

$$\frac{1}{2}\left(1 - \frac{1}{e}\right) + \frac{1}{2} \cdot \frac{7}{8} \approx .753 \geq \frac{3}{4},$$

so the claim is proven. □

Notice that taking the best 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 best solution of the two derandomized algorithms is a deterministic $\frac{3}{4}$-approximation algorithm.

5.6 Non-linear 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 non-linear 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, we only need analyze the probability that a given clause C_j is satisfied. By the definition of f,

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

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

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

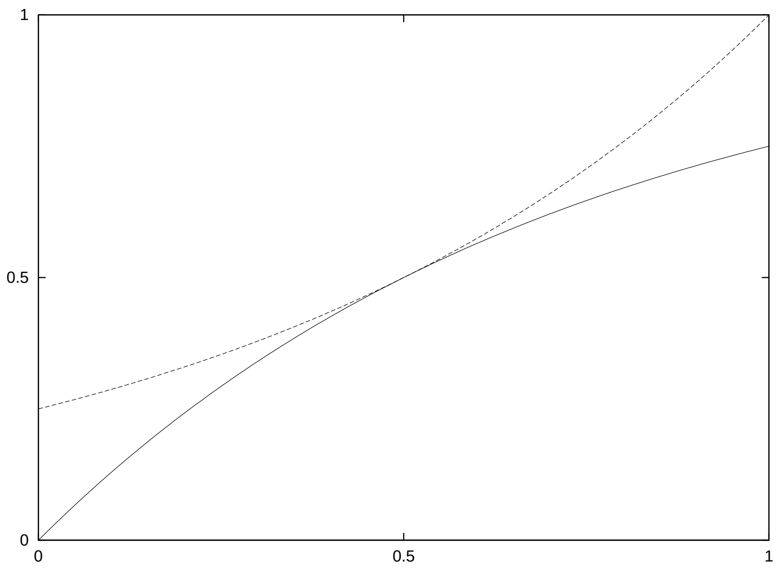

Figure 5.3: A plot of the functions of (5.2). The upper curve is 4^{x-1} and the lower curve is $1 - 4^{-x}$.

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

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

□

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 one 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 value of an optimal solution to the integer programming formulation to value of an optimal solution to its linear programming relaxation.*