14 Rounding Applied to Set Cover

We will introduce the technique of LP-rounding by using it to design two approximation algorithms for the set cover problem, Problem 2.1. The first is a simple rounding algorithm achieving a guarantee of f, where f is the frequency of the most frequent element. The second algorithm, achieving an approximation guarantee of $O(\log n)$, illustrates the use of randomization in rounding.

Consider the polyhedron defined by feasible solutions to an LP-relaxation. For some problems, one can find special properties of extreme point solutions of this polyhedron, which can yield rounding-based algorithms. One such property is *half-integrality*, i.e., in each extreme point solution, every coordinate is $0, 1$, or $1/2$. In Section 14.3 we will show that the vertex cover problem possesses this remarkable property. This directly gives a factor 2 algorithm for weighted vertex cover; namely, find an optimal extreme point solution and round all the halves to 1. A more general property, together with an enhanced rounding algorithm, called iterated rounding, is introduced in Chapter 23.

14.1 A simple rounding algorithm

A linear programming relaxation for the set cover problem is given in LP(13.2). One way of converting a solution to this linear program into an integral solution is to round up all nonzero variables to 1. It is easy to construct examples showing that this could increase the cost by a factor of $\Omega(n)$ (see Example 14.3). However, this simple algorithm does achieve the desired approximation guarantee of f (see Exercise 14.1). Let us consider a slight modification of this algorithm that is easier to prove and picks fewer sets in general:

Algorithm 14.1 (Set cover via LP-rounding)

```
1. Find an optimal solution to the LP-relaxation.
2. Pick all sets $S$ for which $x_S \geq 1/f$ in this solution.
```

Theorem 14.2 *Algorithm 14.1 achieves an approximation factor of f for the set cover problem.*

Proof: Let $\mathcal{C}$ be the collection of picked sets. Consider an arbitrary element e. Since e is in at most f sets, one of these sets must be picked to the extent of at least $1/f$ in the fractional cover. Thus, e is covered by $\mathcal{C}$, and hence $\mathcal{C}$ is a valid set cover. The rounding process increases x_S, for each set $S \in \mathcal{C}$, by a factor of at most f. Therefore, the cost of $\mathcal{C}$ is at most f times the cost of the fractional cover, thereby proving the desired approximation guarantee. □

The set cover instance arising from a vertex cover problem has $f = 2$. Therefore, Algorithm 14.1 gives a factor 2 approximation algorithm for the weighted vertex cover problem, thus matching the approximation guarantee established in Theorem 2.7.

Example 14.3 Let us give a tight example for Algorithm 14.1. For simplicity, we will view a set cover instance as a hypergraph: sets correspond to vertices and elements correspond to hyperedges (this is a generalization of the transformation that helped us view a set cover instance with each element having frequency 2 as a vertex cover instance).

Let $V_1, \ldots, V_k$ be k disjoint sets of cardinality n each. The hypergraph has vertex set $V = V_1 \cup \ldots \cup V_k$, and n^k hyperedges; each hyperedge picks one vertex from each V_i. In the set cover instance, elements correspond to hyperedges and sets correspond to vertices. Once again, inclusion corresponds to incidence. Each set has cost 1. Picking each set to the extent of $1/k$ gives an optimal fractional cover of cost n. Given this fractional solution, the rounding algorithm will pick all nk sets. On the other hand, picking all sets corresponding to vertices in V_1 gives a set cover of cost n. □

14.2 Randomized rounding

A natural idea for rounding an optimal fractional solution is to view the fractions as probabilities, flip coins with these biases and round accordingly. Let us show how this idea leads to an $O(\log n)$ factor randomized approximation algorithm for the set cover problem.

First, we will show that each element is covered with constant probability by the sets picked by this process. Repeating this process $O(\log n)$ times, and picking a set if it is chosen in any of the iterations, we get a set cover with high probability, by a standard coupon collector argument. The expected cost of cover picked in this manner is $O(\log n) \cdot \mathrm{OPT}_f \leq O(\log n) \cdot \mathrm{OPT}$, where OPT_f is the cost of an optimal solution to the LP-relaxation. Applying Markov's Inequality, we convert this into a high probability statement. We provide details below.

Let $\boldsymbol{x} = \boldsymbol{p}$ be an optimal solution to the linear program. For each set $S \in \mathcal{S}$, pick S with probability p_S, the entry corresponding to S in $\boldsymbol{p}$. Let $\mathcal{C}$ be the collection of sets picked. The expected cost of $\mathcal{C}$,

$$\mathbf{E}[\text{cost}(\mathcal{C})] = \sum_{S \in \mathcal{S}} \mathbf{Pr}[S \text{ is picked}] \cdot c_S = \sum_{S \in \mathcal{S}} p_S \cdot c_S = \text{OPT}_f .$$

Next, let us compute the probability that an element $a \in U$ is covered by $\mathcal{C}$. Suppose that a occurs in k sets of $\mathcal{S}$. Let the probabilities associated with these sets be $p_1, \ldots, p_k$. Since a is fractionally covered in the optimal solution, $p_1 + p_2 + \cdots + p_k \geq 1$. Using elementary calculus, it is easy to show that under this condition, the probability that a is covered by $\mathcal{C}$ is minimized when each of the p_i's is $1/k$. Thus,

$$\mathbf{Pr}[a \text{ is covered by } \mathcal{C}] \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e},$$

where e is the base of natural logarithms. Hence each element is covered with constant probability by $\mathcal{C}$.

To get a complete set cover, independently pick $c \log n$ such subcollections, and compute their union, say $\mathcal{C}'$, where c is a constant such that

$$\left(\frac{1}{e}\right)^{c \log n} \leq \frac{1}{4n}.$$

Now,

$$\mathbf{Pr}[a \text{ is not covered by } \mathcal{C}'] \leq \left(\frac{1}{e}\right)^{c \log n} \leq \frac{1}{4n}.$$

Summing over all elements $a \in U$, we get

$$\mathbf{Pr}[\mathcal{C}' \text{ is not a valid set cover}] \leq n \cdot \frac{1}{4n} \leq \frac{1}{4}.$$

Clearly, $\mathbf{E}[\mathcal{C}'] \leq \text{OPT}_f \cdot c \log n$. Applying Markov's Inequality (see Section B.2) with $t = \text{OPT}_f \cdot 4c \log n$, we get

$$\mathbf{Pr}[\text{cost}(\mathcal{C}') \geq \text{OPT}_f \cdot 4c \log n] \leq \frac{1}{4}.$$

The probability of the union of the two undesirable events is $\leq 1/2$. Hence,

$$\mathbf{Pr}[\mathcal{C}' \text{ is a valid set cover and has cost} \leq \text{OPT}_f \cdot 4c \log n] \geq \frac{1}{2}.$$

Observe that we can verify in polynomial time whether $\mathcal{C}'$ satisfies both these conditions. If not, we repeat the entire algorithm. The expected number of repetitions needed at most 2.

14.3 Half-integrality of vertex cover

Consider the vertex cover problem with arbitrary weights. Let $c : V \to \mathbf{Q}^+$ be the function assigning nonnegative weights to the vertices. The integer program for this problem is:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{v \in V} c(v) x_v & \\ \text{subject to} & x_u + x_v \geq 1, & (u,v) \in E \\ & x_v \in \{0,1\}, & v \in V \end{array} \tag{14.1}$$

The LP-relaxation of this integer program is:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{v \in V} c(v) x_v & \\ \text{subject to} & x_u + x_v \geq 1, & (u,v) \in E \\ & x_v \geq 0, & v \in V \end{array} \tag{14.2}$$

Recall that an *extreme point solution* of a set of linear inequalities is a feasible solution that cannot be expressed as convex combination of two other feasible solutions. A *half-integral solution* to LP (14.2) is a feasible solution in which each variable is 0, 1, or 1/2.

Lemma 14.4 *Let $\boldsymbol{x}$ be a feasible solution to LP (14.2) that is not half-integral. Then, $\boldsymbol{x}$ is the convex combination of two feasible solutions and is therefore not an extreme point solution for the set of inequalities in LP (14.2).*

Proof: Consider the set of vertices for which solution $\boldsymbol{x}$ does not assign half-integral values. Partition this set as follows.

$$V_+ = \left\{ v \,\middle|\, \frac{1}{2} < x_v < 1 \right\}, \qquad V_- = \left\{ v \,\middle|\, 0 < x_v < \frac{1}{2} \right\}.$$

For $\varepsilon > 0$, define the following two solutions.

$$y_v = \begin{cases} x_v + \varepsilon, & x_v \in V_+ \\ x_v - \varepsilon, & x_v \in V_- \\ \quad x_v, \textit{otherwise} \end{cases}, \qquad z_v = \begin{cases} x_v - \varepsilon, & x_v \in V_+ \\ x_v + \varepsilon, & x_v \in V_- \\ \quad x_v, \textit{otherwise.} \end{cases}$$

By assumption, $V_+ \cup V_- \neq \emptyset$, and so $\boldsymbol{x}$ is distinct from $\boldsymbol{y}$ and $\boldsymbol{z}$. Furthermore, $\boldsymbol{x}$ is a convex combination of $\boldsymbol{y}$ and $\boldsymbol{z}$, since $\boldsymbol{x} = \frac{1}{2}(\boldsymbol{y} + \boldsymbol{z})$. We will show, by choosing $\varepsilon > 0$ small enough, that $\boldsymbol{y}$ and $\boldsymbol{z}$ are both feasible solutions for LP (14.2), thereby establishing the lemma.

Ensuring that all coordinates of $\boldsymbol{y}$ and $\boldsymbol{z}$ are nonnegative is easy. Next, consider the edge constraints. Suppose $x_u + x_v > 1$. Clearly, by choosing ε small enough, we can ensure that $\boldsymbol{y}$ and $\boldsymbol{z}$ do not violate the constraint for such an edge. Finally, consider an edge such that $x_u + x_v = 1$. There are essentially three possibilities for x_u and x_v. $x_u = x_v = \frac{1}{2}$; $x_u = 0, x_v = 1$; and $u \in V_+, v \in V_-$. In all three cases, for any choice of ε,

$$x_u + x_v = y_u + y_v = z_u + z_v = 1.$$

The lemma follows. □

This leads to:

Theorem 14.5 *Any extreme point solution for the set of inequalities in LP (14.2) is half-integral.*

Theorem 14.5 directly leads to a factor 2 approximation algorithm for weighted vertex cover: find an extreme point solution, and pick all vertices that are set to half or one in this solution.

14.4 Exercises

14.1 Modify Algorithm 14.1 so that it picks all sets that are nonzero in the fractional solution. Show that the algorithm also achieves a factor of f.
Hint: Use the primal complementary slackness conditions to prove this.

14.2 Consider the collection of sets, $\mathcal{C}$, picked by the randomized rounding algorithm. Show that with some constant probability, $\mathcal{C}$ covers at least half the elements at a cost of at most $O(\text{OPT})$.

14.3 Give $O(\log n)$ factor randomized rounding algorithms for the set multicover and multiset multicover problems (see Section 13.2).

14.4 Give a (non-bipartite) tight example for the half-integrality-based algorithm for weighted vertex cover.

14.5 (J. Cheriyan) Give a polynomial time algorithm for the following problem. Given a graph G with nonnegative vertex weights and a valid, though not necessarily optimal, coloring of G, find a vertex cover of weight $\leq (2 - \frac{2}{k})\text{OPT}$, where k is the number of colors used.

14.6 Give a counterexample to the following claim. A set cover instance in which each element is in exactly f sets has a $(1/f)$-integral optimal fractional solution (i.e., in which each set is picked an integral multiple of $1/f$).

14.7 This exercise develops a combinatorial algorithm for finding an optimal half integral vertex cover. Given undirected graph $G = (V, E)$ and a non-negative cost function c on vertices, obtain bipartite graph $H(V', V'', E')$ as follows. Corresponding to each vertex $v \in V$, there is a vertex $v' \in V'$ and $v'' \in V''$ each of cost $c(v)/2$. Corresponding to each edge $(u, v) \in E$, there are two edges $(u', v''), (u'', v') \in E'$. Show that a vertex cover in H can be mapped to a half-integral vertex cover in G preserving total cost and vice versa. Use the fact that an optimal vertex cover in a bipartite graph can be found in polynomial time to obtain an optimal half-integral vertex cover in G.

14.8 Consider LP (12.8), introduced in Exercise 12.7, for a non-bipartite graph $G = (V, E)$.

1. Show that it is not an exact relaxation for the maximum matching problem in G.
2. Show that this LP always has a half-integral optimal solution.

14.9 In an attempt to improve the running time of the algorithm obtained in Exercise 9.7 for bin packing, consider going to the LP-relaxation of the integer programming and using LP-rounding. What guarantee can you establish for bin packing through this method?

14.5 Notes

Algorithm 14.1 is due to Hochbaum [125]. For a more sophisticated randomized rounding algorithm for set cover, see Srinivasan [244]. Theorem 14.5 is due to Nemhauser and Trotter [213].

16 Maximum Satisfiability

The maximum satisfiability problem has been a classical problem in approximation algorithms. More recently, its study has led to crucial insights in the area of hardness of approximation (see Chapter 29). In this chapter, we will use LP-rounding, with randomization, to obtain a 3/4 factor approximation algorithm. We will derandomize this algorithm using the *method of conditional expectation.*

Problem 16.1 (Maximum satisfiability (MAX-SAT)) Given a conjunctive normal form formula f on Boolean variables $x_1, \ldots, x_n$, and nonnegative weights, w_c, for each clause c of f, find a truth assignment to the Boolean variables that maximizes the total weight of satisfied clauses. Let $\mathcal{C}$ represent the set of clauses of f, i.e., $f = \bigwedge_{c \in \mathcal{C}} c$. Each clause is a disjunction of literals; each literal being either a Boolean variable or its negation. Let size(c) denote the *size* of clause c, i.e., the number of literals in it. We will assume that the sizes of clauses in f are arbitrary.

For any positive integer k, we will denote by MAX-kSAT the restriction of MAX-SAT to instances in which each clause is of size at most k. MAX-SAT is **NP**-hard; in fact, even MAX-2SAT is **NP**-hard (in contrast, 2SAT is in **P**). We will first present two approximation algorithms for MAX-SAT, having guarantees of $1/2$ and $1-1/e$, respectively. The first performs better if the clause sizes are large, and the seconds performs better if they are small. We will then show how an appropriate combination of the two algorithms achieves the promised approximation guarantee.

In the interest of minimizing notation, let us introduce common terminology for all three algorithms. Random variable W will denote the total weight of satisfied clauses. For each clause c, random variable W_c denotes the weight contributed by clause c to W. Thus, $W = \sum_{c \in \mathcal{C}} W_c$ and

$$\mathbf{E}[W_c] = w_c \cdot \mathbf{Pr}[c \text{ is satisfied}].$$

(Strictly speaking, this is abuse of notation, since the randomization used by the three algorithms is different.)

16.1 Dealing with large clauses

The first algorithm is straightforward. Set each Boolean variable to be True independently with probability 1/2 and output the resulting truth assignment, say τ. For $k \geq 1$, define $\alpha_k = 1 - 2^{-k}$.

Lemma 16.2 *If* size$(c) = k$, *then* $\mathbf{E}[W_c] = \alpha_k w_c$.

Proof: Clause c is not satisfied by τ iff all its literals are set to False. The probability of this event is 2^{-k}. □

For $k \geq 1, \alpha_k \geq 1/2$. By linearity of expectation,

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geq \frac{1}{2} \sum_{c \in \mathcal{C}} w_c \geq \frac{1}{2}\mathrm{OPT},$$

where we have used a trivial upper bound on OPT – the total weight of clauses in $\mathcal{C}$.

Instead of converting this into a high probability statement, with a corresponding loss in guarantee, we show how to derandomize this procedure. The resulting algorithm deterministically computes a truth assignment such that the weight of satisfied clauses is $\geq \mathbf{E}[W] \geq \mathrm{OPT}/2$.

Observe that α_k increases with k and the guarantee of this algorithm is 3/4 if each clause has two or more literals. (The next algorithm is designed to deal with unit clauses more effectively.)

16.2 Derandomizing via the method of conditional expectation

We will critically use the self-reducibility of SAT (see Section A.5). Consider the self-reducibility tree T for formula f. Each internal node at level i corresponds to a setting for Boolean variables $x_1, \ldots, x_i$, and each leaf represents a complete truth assignment to the n variables. Let us label each node of T with its conditional expectation as follows. Let $a_1, \ldots, a_i$ be a truth assignment to $x_1, \ldots, x_i$. The node corresponding to this assignment will be labeled with $\mathbf{E}[W | x_1 = a_1, \ldots, x_i = a_i]$. If $i = n$, this is a leaf node and its conditional expectation is simply the total weight of clauses satisfied by its truth assignment.

Lemma 16.3 *The conditional expectation of any node in T can be computed in polynomial time.*

Proof: Consider a node $x_1 = a_1, \ldots, x_i = a_i$. Let ϕ be the Boolean formula, on variables $x_{i+1}, \ldots, x_n$, obtained for this node via self-reducibility. Clearly, the expected weight of satisfied clauses of ϕ under a random truth assignment to the variables $x_{i+1}, \ldots, x_n$ can be computed in polynomial time. Adding to this the total weight of clauses of f already satisfied by the partial assignment $x_1 = a_1, \ldots, x_i = a_i$ gives the answer. □

Theorem 16.4 *We can compute, in polynomial time, a path from the root to a leaf such that the conditional expectation of each node on this path is* $\geq \mathbf{E}[W]$.

Proof: The conditional expectation of a node is the average of the conditional expectations of its two children, i.e.,

$$\begin{aligned}\mathbf{E}[W|x_1 = a_1, ..., x_i = a_i] = {} & \mathbf{E}[W|x_1 = a_1, ..., x_i = a_i, x_{i+1} = \text{True}]/2 + \\ & \mathbf{E}[W|x_1 = a_1, ..., x_i = a_i, x_{i+1} = \text{False}]/2.\end{aligned}$$

The reason, of course, is that x_{i+1} is equally likely to be set to True or False. As a result, the child with the larger value has a conditional expectation at least as large as that of the parent. This establishes the existence of the desired path. As a consequence of Lemma 16.3, it can be computed in polynomial time. □

The deterministic algorithm follows as a corollary of Theorem 16.4. We simply output the truth assignment on the leaf node of the path computed. The total weight of clauses satisfied by it is $\geq \mathbf{E}[W]$.

Let us show that the technique outlined above can, in principle, be used to derandomize more complex randomized algorithms. Suppose the algorithm does not set the Boolean variables independently of each other (for instance, see Remark 16.6). Now,

$$\begin{aligned}&\mathbf{E}[W|x_1 = a_1, ..., x_i = a_i] = \\ &\mathbf{E}[W|x_1 = a_1, ..., x_i = a_i, x_{i+1} = \text{True}] \cdot \mathbf{Pr}[x_{i+1} = \text{True}|x_1 = a_1, ..., x_i = a_i] + \\ &\mathbf{E}[W|x_1 = a_1, ..., x_i = a_i, x_{i+1} = \text{False}] \cdot \mathbf{Pr}[x_{i+1} = \text{False}|x_1 = a_1, ..., x_i = a_i].\end{aligned}$$

The sum of the two conditional probabilities is again 1, since the two events are exhaustive. So, the conditional expectation of the parent is still a convex combination of the conditional expectations of the two children. If we can determine, in polynomial time, which of the two children has a larger value, we can again derandomize the algorithm. However, computing the conditional expectations may not be easy. Observe how critically independence was used in the proof of Lemma 16.3. It was because of independence that we could assume a random truth assignment on Boolean variables $x_{i+1}, \ldots, x_n$ and thereby compute the expected weight of satisfied clauses of ϕ.

In general, a randomized algorithm may pick from a larger set of choices and not necessarily with equal probability. But once again a convex combination of the conditional expectations of these choices, given by the probabilities of picking them, equals the conditional expectation of the parent. Hence there must be a choice that has at least as large a conditional expectation as the parent.

16.3 Dealing with small clauses via LP-rounding

Following is an integer program for MAX-SAT. For each clause $c \in \mathcal{C}$, let S_c^+ (S_c^-) denote the set of Boolean variables occurring nonnegated (negated) in c. The truth assignment is encoded by $\boldsymbol{y}$. Picking $y_i = 1$ ($y_i = 0$) denotes setting x_i to True (False). The constraint for clause c ensures that z_c can be set to 1 only if at least one of the literals occurring in c is set to True, i.e., if clause c is satisfied by the picked truth assignment.

$$\begin{aligned}
\text{maximize} \quad & \sum_{c \in \mathcal{C}} w_c z_c && (16.1)\\
\text{subject to} \quad & \forall c \in \mathcal{C}: \ \sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c \\
& \forall c \in \mathcal{C}: \ z_c \in \{0, 1\} \\
& \forall i: \ y_i \in \{0, 1\}
\end{aligned}$$

The LP-relaxation is:

$$\begin{aligned}
\text{maximize} \quad & \sum_{c \in \mathcal{C}} w_c z_c && (16.2)\\
\text{subject to} \quad & \forall c \in \mathcal{C}: \ \sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c \\
& \forall c \in \mathcal{C}: \ 1 \geq z_c \geq 0 \\
& \forall i: \ 1 \geq y_i \geq 0
\end{aligned}$$

The algorithm is again straightforward. Solve LP (16.2). Let $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ denote the optimal solution. Independently set x_i to True with probability y_i^*, for $1 \leq i \leq n$. Output the resulting truth assignment, say τ.

We will use the random variables W and W_c defined in Section 16.1. For $k \geq 1$, define

$$\beta_k = 1 - \left(1 - \frac{1}{k}\right)^k .$$

Lemma 16.5 *If* size$(c) = k$, *then*

$$\mathbf{E}[W_c] \geq \beta_k w_c z_c^* .$$

Proof: We may assume w.l.o.g. that all literals in c appear nonnegated (if x_i appears negated, we can replace x_i with $\overline{x}_i$ throughout f and modify LP (16.2) accordingly without affecting z_c^* or W_c). Further, by renaming variables, we may assume $c = (x_1 \vee \ldots \vee x_k)$.

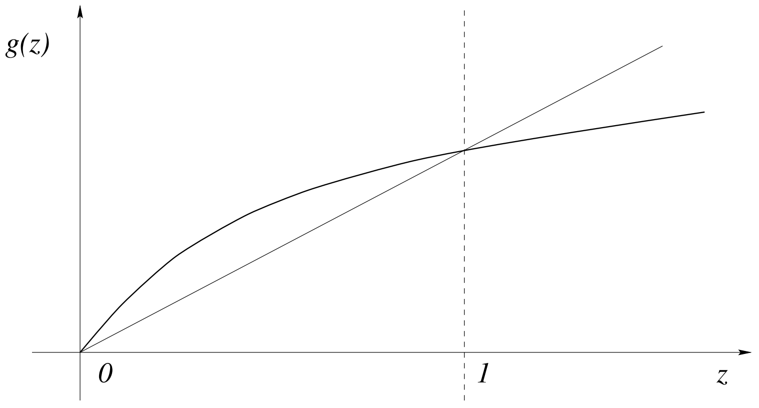
Clause c is satisfied if $x_1, \ldots, x_k$ are not all set to False. The probability of this event is

$$1 - \prod_{i=1}^{k}(1-y_i) \geq 1 - \left(\frac{\sum_{i=1}^{k}(1-y_i)}{k}\right)^k = 1 - \left(1 - \frac{\sum_{i=1}^{k} y_i}{k}\right)^k$$
$$\geq 1 - \left(1 - \frac{z_c^*}{k}\right)^k ,$$

where the first inequality follows from the arithmetic-geometric mean inequality which states that for nonnegative numbers $a_1, \ldots, a_k$,

$$\frac{a_1 + \ldots + a_k}{k} \geq \sqrt[k]{a_1 \times \ldots \times a_k}.$$

The second inequality uses the constraint in LP (16.2) that $y_1 + \ldots + y_k \geq z_c$.



Define function g by:

$$g(z) = 1 - \left(1 - \frac{z}{k}\right)^k .$$

This is a concave function with $g(0) = 0$ and $g(1) = \beta_k$. Therefore, for $z \in [0,1], g(z) \geq \beta_k z$. Hence, $\mathbf{Pr}[c \text{ is satisfied}] \geq \beta_k z_c^*$. The lemma follows. □

Notice that β_k is a decreasing function of k. Thus, if all clauses are of size at most k,

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geq \beta_k \sum_{c \in \mathcal{C}} w_c z_c^* = \beta_k \mathrm{OPT}_f \geq \beta_k \mathrm{OPT},$$

where OPT_f is the optimal solution to LP (16.2). Clearly, $\mathrm{OPT}_f \geq \mathrm{OPT}$. This algorithm can also be derandomized using the method of conditional expectation (Exercise 16.3). Hence, for MAX-SAT instances with clause sizes at most k, it is a β_k factor approximation algorithm. Since

$$\forall k \in \mathbf{Z}^+ : \left(1 - \frac{1}{k}\right)^k > \frac{1}{e},$$

this is a $1 - 1/e$ factor algorithm for MAX-SAT.

16.4 A 3/4 factor algorithm

We will combine the two algorithms as follows. Let b be the flip of a fair coin. If $b = 0$, run the first randomized algorithm, and if $b = 1$, run the second randomized algorithm.

Remark 16.6 Notice that we are effectively setting x_i to True with probability $\frac{1}{4} + \frac{1}{2}y_i^*$; however, the x_i's are *not* set independently!

Let $\boldsymbol{z}^*$ be the optimal solution of LP (16.2) on the given instance.

Lemma 16.7 $\mathbf{E}[W_c] \geq \frac{3}{4} w_c z_c^*.$

Proof: Let size$(c) = k$. By Lemma 16.2,

$$\mathbf{E}[W_c \mid b = 0] = \alpha_k w_c \geq \alpha_k w_c z_c^*,$$

where we have used the fact that $z_c^* \leq 1$. By Lemma 16.5,

$$\mathbf{E}[W_c \mid b = 1] \geq \beta_k w_c z_c^*.$$

Combining we get

$$\mathbf{E}[W_c] = \frac{1}{2}(\mathbf{E}[W_c \mid b = 0] + \mathbf{E}[W_c \mid b = 1]) \geq w_c z_c^* \frac{(\alpha_k + \beta_k)}{2}.$$

Now, $\alpha_1 + \beta_1 = \alpha_2 + \beta_2 = 3/2$, and for $k \geq 3$, $\alpha_k + \beta_k \geq 7/8 + (1 - 1/e) \geq 3/2$. The lemma follows. □

By linearity of expectation,

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geq \frac{3}{4} \sum_{c \in \mathcal{C}} w_c z_c^* = \frac{3}{4}\mathrm{OPT}_f \geq \frac{3}{4}\mathrm{OPT}, \tag{16.3}$$

where OPT_f is the optimal solution to LP (16.2). Finally, consider the following deterministic algorithm.

Algorithm 16.8 (MAX-SAT – factor $3/4$)

1. Use the derandomized factor $1/2$ algorithm to get a truth assignment, τ_1.
2. Use the derandomized factor $1 - 1/e$ algorithm to get a truth assignment, τ_2.
3. Output the better of the two assignments.

Theorem 16.9 *Algorithm 16.8 is a deterministic factor $3/4$ approximation algorithm for MAX-SAT.*

Proof: One of the two conditional expectations, $\mathbf{E}[W \mid b = 0]$ and $\mathbf{E}[W \mid b = 1]$, is at least as large as $\mathbf{E}[W]$. Hence, the total weight of clauses satisfied by the better of τ_1 and τ_2 is at least as large as $\mathbf{E}[W]$. $\square$

By (16.3), $\mathbf{E}[W] \geq \frac{3}{4}\mathrm{OPT}_f$. The weight of the integral solution produced by Algorithm 16.8 is at least $\mathbf{E}[W]$. Therefore, the integrality gap of LP (16.2) is $\geq 3/4$. Below we show that this is tight.

Example 16.10 Consider the SAT formula $f = (x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$, where each clause is of unit weight. It is easy to see that setting $y_i = 1/2$ and $z_c = 1$ for all i and c is an optimal solution to LP (16.2) for any instance having size 2 clauses. Therefore $\mathrm{OPT}_f = 4$. On the other hand $\mathrm{OPT} = 3$, and thus for this instance LP (16.2) has a integrality gap of $4/3$. $\square$

Example 16.11 Let us provide a tight example to Algorithm 16.8. Let $f = (x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee z)$, and let the weights of these three clauses be 1, 1, and $2 + \varepsilon$, respectively. By the remark made in Example 16.10, on this instance the factor $1 - 1/e$ algorithm will set each variable to True with probability $1/2$ and so will be the same as the factor $1/2$ algorithm. During derandomization, suppose variable x is set first. The conditional expectations are $\mathbf{E}[W \mid x = \mathrm{True}] = 3 + \varepsilon/2$ and $\mathbf{E}[W \mid x = \mathrm{False}] = 3 + \varepsilon$. Thus, x will be set to False. But this leads to a total weight of $3 + \varepsilon$, whereas by setting x to True we can get a weight of $4 + \varepsilon$. Clearly, we can get an infinite family of such examples by replicating these 3 clauses with new variables. $\square$

16.5 Exercises

16.1 The algorithm of Section 16.1 achieves an approximation guarantee of α_k if all clauses in the given instance have size at least k. Give a tight example of factor α_k for this algorithm.

16.2 Show that the following is a factor 1/2 algorithm for MAX-SAT. Let τ be an arbitrary truth assignment and τ' be its complement, i.e., a variable is True in τ iff it is False in τ'. Compute the weight of clauses satisfied by τ and τ', then output the better assignment.

16.3 Use the method of conditional expectation to derandomize the $1 - 1/e$ factor algorithm for MAX-SAT.

16.4 Observe that the randomization used in the 3/4 factor algorithm does not set Boolean variables independently of each other. As remarked in Section 16.2, the algorithm can still, in principle, be derandomized using the method of conditional expectation. Devise a way of doing so. Observe that the algorithm obtained is different from Algorithm 16.8.

16.5 (Goemans and Williamson [104]) Instead of using the solution to LP (16.2), y_i^*, as probability of setting x_i to True, consider the more general scheme of using $g(y_i^*)$, for a suitable function g. Can this lead to an improvement over the factor $1 - 1/e$ algorithm?

16.6 Consider the following randomized algorithm for the maximum cut problem, defined in Exercise 2.1. After the initialization step of Algorithm 2.13, each of the remaining vertices is equally likely to go in sets A or B. Show that the expected size of the cut found is at least OPT/2. Show that the derandomization of this algorithm via the method of conditional expectation is precisely Algorithm 2.13.

16.7 Consider the following generalization of the maximum cut problem.

Problem 16.12 (Linear equations over GF[2]) Given m equations over n GF[2] variables, find an assignment for the variables that maximizes the number of satisfied equations.

1. Show that if $m \leq n$, this problem is polynomial time solvable.
2. In general, the problem is **NP**-hard. Give a factor 1/2 randomized algorithm for it, and derandomize using the method of conditional expectation.

16.8 Consider the obvious randomized algorithm for the MAX k-CUT problem, Problem 2.14 in Exercise 2.3, which assigns each vertex randomly to one of the sets $S_1, \ldots, S_k$. Show that the expected number of edges running between these sets is at least OPT/2. Show that the derandomization of this algorithm, via the method of conditional expectation, gives the greedy algorithm sought in Exercise 2.3.

16.9 Repeat Exercise 16.8 for the maximum directed cut problem, Problem 2.15 in Exercise 2.4, i.e., give a factor 1/4 randomized algorithm, and show that its derandomization gives a greedy algorithm.

16.6 Notes

The factor 1/2 algorithm, which was also the first approximation algorithm for MAX-SAT, is due to Johnson [150]. The first factor 3/4 algorithm was due to Yannakakis [261]. The (simpler) algorithm given here is due to Goemans and Williamson [104]. The method of conditional expectation is implicit in Erdös and Selfridge [74]. Its use for obtaining polynomial time algorithms was pointed out by Spencer [243] (see Raghavan [225] and Alon and Spencer [6] for enhancements to this technique).