$$Y_i = \mathrm{I}\{\text{clause } i \text{ is satisfied}\} ,$$

so that $Y_i = 1$ as long as at least one of the literals in the ith clause is set to 1. Since no literal appears more than once in the same clause, and since we assume that no variable and its negation appear in the same clause, the settings of the three literals in each clause are independent. A clause is not satisfied only if all three of its literals are set to 0, and so $\Pr\{\text{clause } i \text{ is not satisfied}\} = (1/2)^3 = 1/8$. Thus, we have $\Pr\{\text{clause } i \text{ is satisfied}\} = 1 - 1/8 = 7/8$, and Lemma 5.1 on page 130 gives $\mathrm{E}[Y_i] = 7/8$. Let Y be the number of satisfied clauses overall, so that $Y = Y_1 + Y_2 + \cdots + Y_m$. Then, we have

$$\begin{aligned}
\mathrm{E}[Y] &= \mathrm{E}\left[\sum_{i=1}^{m} Y_i\right] \\
&= \sum_{i=1}^{m} \mathrm{E}[Y_i] \qquad \text{(by linearity of expectation)} \\
&= \sum_{i=1}^{m} 7/8 \\
&= 7m/8 .
\end{aligned}$$

Since m is an upper bound on the number of satisfied clauses, the approximation ratio is at most $m/(7m/8) = 8/7$. ■

Approximating weighted vertex cover using linear programming

The ***minimum-weight vertex-cover problem*** takes as input an undirected graph $G = (V, E)$ in which each vertex $v \in V$ has an associated positive weight $w(v)$. The weight $w(V')$ of a vertex cover $V' \subseteq V$ is the sum of the weights of its vertices: $w(V') = \sum_{v \in V'} w(v)$. The goal is to find a vertex cover of minimum weight.

The approximation algorithm for unweighted vertex cover from Section 35.1 won't work here, because the solution it returns could be far from optimal for the weighted problem. Instead, we'll first compute a lower bound on the weight of the minimum-weight vertex cover, by using a linear program. Then we'll "round" this solution and use it to obtain a vertex cover.

Start by associating a variable $x(v)$ with each vertex $v \in V$, and require that $x(v)$ equals either 0 or 1 for each $v \in V$. The vertex cover includes v if and only if $x(v) = 1$. Then the constraint that for any edge (u, v), at least one of u and v must belong to the vertex cover can be expressed as $x(u) + x(v) \geq 1$. This view gives rise to the following ***0-1 integer program*** for finding a minimum-weight vertex cover:

$$\text{minimize} \quad \sum_{v \in V} w(v)\, x(v) \tag{35.12}$$

subject to

$$x(u) + x(v) \geq 1 \qquad \text{for each } (u, v) \in E \tag{35.13}$$

$$x(v) \in \{0, 1\} \quad \text{for each } v \in V\,. \tag{35.14}$$

In the special case in which all the weights $w(v)$ equal 1, this formulation is the optimization version of the NP-hard vertex-cover problem. Let's remove the constraint that $x(v) \in \{0, 1\}$ and replace it by $0 \leq x(v) \leq 1$, resulting in the following linear program:

$$\text{minimize} \quad \sum_{v \in V} w(v)\, x(v) \tag{35.15}$$

subject to

$$x(u) + x(v) \geq 1 \quad \text{for each } (u, v) \in E \tag{35.16}$$

$$x(v) \leq 1 \quad \text{for each } v \in V \tag{35.17}$$

$$x(v) \geq 0 \quad \text{for each } v \in V\,. \tag{35.18}$$

We refer to this linear program as the ***linear-programming relaxation***. Any feasible solution to the 0-1 integer program in lines (35.12)–(35.14) is also a feasible solution to its linear-programming relaxation in lines (35.15)–(35.18). Therefore, the value of an optimal solution to the linear-programming relaxation provides a lower bound on the value of an optimal solution to the 0-1 integer program, and hence a lower bound on the optimal weight in the minimum-weight vertex-cover problem.

The procedure APPROX-MIN-WEIGHT-VC on the facing page starts with a solution to the linear-programming relaxation and uses it to construct an approximate solution to the minimum-weight vertex-cover problem. The procedure works as follows. Line 1 initializes the vertex cover to be empty. Line 2 formulates the linear-programming relaxation in lines (35.15)–(35.18) and then solves this linear program. An optimal solution gives each vertex v an associated value $\bar{x}(v)$, where $0 \leq \bar{x}(v) \leq 1$. The procedure uses this value to guide the choice of which vertices to add to the vertex cover C in lines 3–5: the vertex cover C includes vertex v if and only if $\bar{x}(v) \geq 1/2$. In effect, the procedure "rounds" each fractional variable in the solution to the linear-programming relaxation to either 0 or 1 in order to obtain a solution to the 0-1 integer program in lines (35.12)–(35.14). Finally, line 6 returns the vertex cover C.

Theorem 35.6

Algorithm APPROX-MIN-WEIGHT-VC is a polynomial-time2-approximation algorithm for the minimum-weight vertex-cover problem.

```
APPROX-MIN-WEIGHT-VC(G, w)
1  C = ∅
2  compute x̄, an optimal solution to the linear-programming relaxation
       in lines (35.15)–(35.18)
3  for each vertex v ∈ V
4      if x̄(v) ≥ 1/2
5          C = C ∪ {v}
6  return C
```

Proof Because there is a polynomial-time algorithm to solve the linear program in line 2, and because the **for** loop of lines 3–5 runs in polynomial time, APPROX-MIN-WEIGHT-VC is a polynomial-time algorithm.

It remains to show that APPROX-MIN-WEIGHT-VC is a2-approximation algorithm. Let C^* be an optimal solution to the minimum-weight vertex-cover problem, and let z^* be the value of an optimal solution to the linear-programming relaxation in lines (35.15)–(35.18). Since an optimal vertex cover is a feasible solution to the linear-programming relaxation, z^* must be a lower bound on $w(C^*)$, that is,

$$z^* \le w(C^*) . \tag{35.19}$$

Next, we claim that rounding the fractional values of the variables $\bar{x}(v)$ in lines 3–5 produces a set C that is a vertex cover and satisfies $w(C) \le 2z^*$. To see that C is a vertex cover, consider any edge $(u, v) \in E$. By constraint (35.16), we know that $x(u) + x(v) \ge 1$, which implies that at least one of $\bar{x}(u)$ and $\bar{x}(v)$ is at least $1/2$. Therefore, at least one of u and v is included in the vertex cover, and so every edge is covered.

Now we consider the weight of the cover. We have

$$\begin{aligned}
z^* &= \sum_{v \in V} w(v)\,\bar{x}(v) \\
&\ge \sum_{v \in V : \bar{x}(v) \ge 1/2} w(v)\,\bar{x}(v) \\
&\ge \sum_{v \in V : \bar{x}(v) \ge 1/2} w(v) \cdot \frac{1}{2} \\
&= \sum_{v \in C} w(v) \cdot \frac{1}{2} \\
&= \frac{1}{2} \sum_{v \in C} w(v) \\
&= \frac{1}{2} w(C) .
\end{aligned} \tag{35.20}$$

Combining inequalities (35.19) and (35.20) gives

$$w(C) \le 2z^* \le 2w(C^*) ,$$

and hence APPROX-MIN-WEIGHT-VC is a2-approximation algorithm. ■

Exercises

35.4-1
Show that even if a clause is allowed to contain both a variable and its negation, randomly setting each variable to 1 with probability $1/2$ and to 0 with probability $1/2$ still yields a randomized $8/7$-approximation algorithm.

35.4-2
The ***MAX-CNF satisfiability problem*** is like the MAX-3-CNF satisfiability problem, except that it does not restrict each clause to have exactly three literals. Give a randomized 2-approximation algorithm for the MAX-CNF satisfiability problem.

35.4-3
In the MAX-CUT problem, the input is an unweighted undirected graph $G = (V, E)$. We define a cut $(S, V - S)$ as in Chapter 21 and the ***weight*** of a cut as the number of edges crossing the cut. The goal is to find a cut of maximum weight. Suppose that each vertex v is randomly and independently placed into S with probability $1/2$ and into $V - S$ with probability $1/2$. Show that this algorithm is a randomized 2-approximation algorithm.

35.4-4
Show that the constraints in line (35.17) are redundant in the sense that removing them from the linear-programming relaxation in lines (35.15)–(35.18) yields a linear program for which any optimal solution x must satisfy $x(v) \le 1$ for each $v \in V$.

35.5 The subset-sum problem

Recall from Section 34.5.5 that an instance of the subset-sum problem is given by a pair (S, t), where S is a set $\{x_1, x_2, \ldots, x_n\}$ of positive integers and t is a positive integer. This decision problem asks whether there exists a subset of S that adds up exactly to the target value t. As we saw in Section 34.5.5, this problem is NP-complete.

The optimization problem associated with this decision problem arises in practical applications. The optimization problem seeks a subset of $\{x_1, x_2, \ldots, x_n\}$