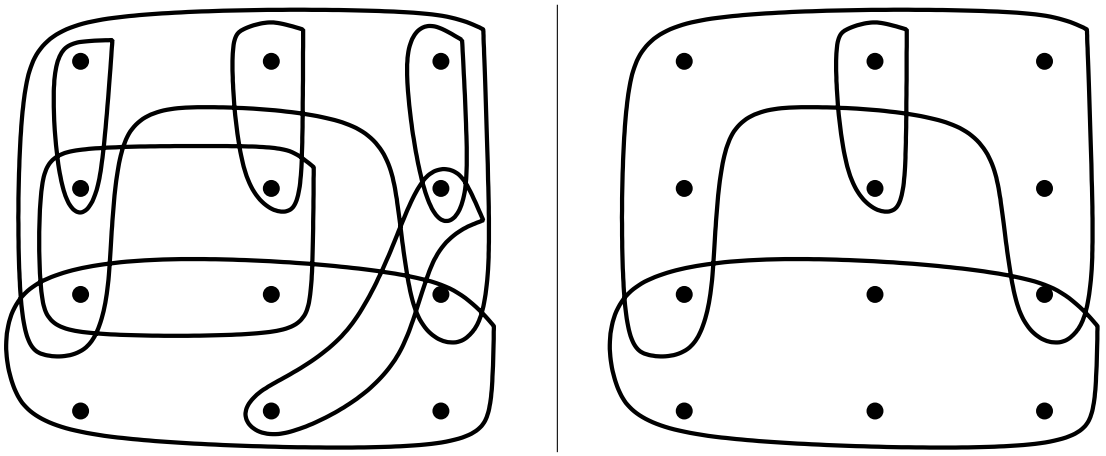

INPUT OUTPUT

18.1 Set Cover

Input description: A collection of subsets $S = \{S_1, \ldots, S_m\}$ of the universal set $U = \{1, \ldots, n\}$.

Problem description: What is the smallest subset T of S whose union equals the universal set—i.e. , $\cup_{i=1}^{|T|} T_i = U$?

Discussion: Set cover arises when you try to efficiently acquire items that have been packaged in a fixed set of lots. You seek a collection of at least one of each distinct type of item, while buying as few lots as possible. Finding *a* set cover is easy, because you can always buy one of each possible lot. However, identifying a small set cover let you do the same job for less money. Set cover provided a natural formulation of the Lotto ticket optimization problem discussed in Section 1.6 (page 23). There we seek to buy the smallest number of tickets needed to cover all of a given set of combinations.

Boolean logic minimization is another interesting application of set cover. We are given a particular Boolean function of k variables, which describes whether the desired output is 0 or 1 for each of the 2^k possible input vectors. We seek the simplest circuit that exactly implements this function. One approach is to find a disjunctive normal form (DNF) formula on the variables and their complements, such as $x_1\bar{x}_2 + \bar{x}_1\bar{x}_2$. We could build one "and" term for each input vector and then "or" them all together, but we might save considerably by factoring out common subsets of variables. Given a set of feasible "and" terms, each of which covers a

subset of the vectors we need, we seek to "or" together the smallest number of terms that realize the function. This is exactly the set cover problem.

There are several variations of set cover problems to be aware of:

- *Are you allowed to cover elements more than once?* – The distinction here is between *set cover* and *set packing*, which is discussed in Section 18.2 (page 625). We should take advantage of the freedom to cover elements multiple times if we have it, as it usually results in a smaller covering.

- *Are your sets derived from the edges or vertices of a graph?* – Set cover is a very general problem, and includes several useful graph problems as special cases. Suppose instead that you seek the smallest set of edges in a graph that will cover each vertex at least once. The solution is the maximum *matching* in the graph (see Section 15.6 (page 498)), plus arbitrary edges to cover any unmatched vertices. Now suppose instead that you seek the smallest set of vertices that cover each edge at least once. This is the *vertex cover* problem, discussed in Section 16.3 (page 530).

  It is instructive to show how to model vertex cover as an instance of set cover. Let the universal set U correspond to the set of edges $\{e_1, \ldots, e_m\}$. Construct n subsets, with S_i consisting of the edges incident on vertex v_i. Although vertex cover is just an instance of set cover in disguise, you should take advantage of the superior heuristics that exist for the more restricted vertex cover problem.

- *Do your subsets contain only two elements each?* – You are in luck if all of your subsets have at most two elements each. This special case can be solved efficiently to optimality because it reduces to finding a maximum matching in a graph. Unfortunately, the problem becomes NP-complete as soon as your subsets have three elements each.

- *Do you want to cover elements with sets, or sets with elements?* – In the *hitting set* problem, we seek a small number of items that together represent each subset in a given population. Hitting set is illustrated in Figure 18.1. The input is identical to set cover, but instead we seek the smallest subset of elements $T \subset U$ such that each subset S_i contains at least one element of T. Thus, $S_i \cap T \neq \emptyset$ for all $1 \leq i \leq m$. Suppose we desire a small Congress with at least one representative for each ethnic group. If each ethnic group is defined by a subset of people, the minimum hitting set gives the smallest possible politically correct Congress.

  Hitting set is *dual* to set cover, meaning that it is exactly the same problem in disguise. Replace each element of U by a set of the names of the subsets that contain it. Now S and U have exchanged roles, for we seek a set of subsets from U to cover all the elements of S. This is exactly set cover, so we can use any set cover code to solve hitting set after performing this simple translation. See Figure 18.1 for an example.

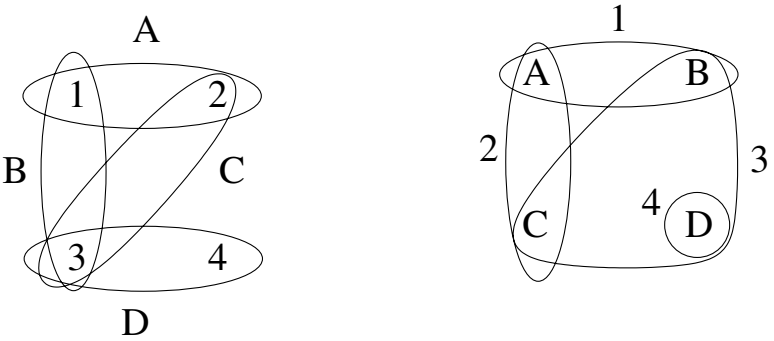

Figure 18.1: A hitting set instance optimally solved by selecting elements 1 and 3 or 2 and 3 (l). This problem converted to a dual set cover instance optimally solved by selecting subsets 1 and 3 or 2 and 4 (r).

Set cover must be at least as hard as vertex cover, so it is also NP-complete. In fact, it is somewhat harder. Approximation algorithms do no worse than twice optimal for vertex cover, but the best approximation algorithm for set cover is $\Theta(\lg n)$ times optimal.

Greedy is the most natural and effective heuristic for set cover. Begin by selecting the largest subset for the cover, and then delete all its elements from the universal set. We add the subset containing the largest number of remaining uncovered elements repeatedly until all are covered. This heuristic always gives a set cover using at most $\ln n$ times as many sets as optimal. In practice it usually does a lot better.

The simplest implementation of the greedy heuristic sweeps through the entire input instance of m subsets for each greedy step. However, by using such data structures as linked lists and a bounded-height priority queue (see Section 12.2 (page 373)), the greedy heuristic can be implemented in $O(S)$ time, where $S = \cup_{i=1}^{m} |S_i|$ is the size of the input representation.

It pays to check whether or not there exist elements that exist in only a few subsets—ideally only one. If so, we should select the biggest subsets containing these elements at the very beginning. We must take such a subset eventually, and they carry along other elements that we might have paid extra to cover if we wait until later.

Simulated annealing is likely to produce somewhat better set covers than these simple heuristics. Backtracking can be used to guarantee you an optimal solution, but it is usually not worth the computational expense.

An alternate and more powerful approach rests on the integer programming formulation of set cover. Let the integer 0-1 variable s_i denote whether subset S_i is selected for a given cover. Each universal set element x adds the constraint

$$\sum_{x \in S_i} s_i \geq 1$$

to ensure that it is covered by at least one selected subset. The minimum set cover satisfies all constraints while minimizing $\sum_i s_i$. This integer program can be easily generalized to weighted set cover (allowing nonuniform costs for different subsets. Relaxing this to a linear program (i.e. , allowing $0 \leq s_i \leq 1$ instead of constricting each variable to be either 0 or 1) allows efficient and effective heuristics using rounding techniques.

Implementations: Both the greedy heuristic and the above ILP formulation are sufficiently simple in their respective worlds that one has to implement them from scratch.

Pascal implementations of an exhaustive search algorithm for set packing, as well as heuristics for set cover, appear in [SDK83]. See Section 19.1.10 (page 662).

SYMPHONY is a mixed-integer linear programming solver that includes a set partitioning solver. It is available at *http://branchandcut.org/SPP/*

Notes: An old but classic survey article on set cover is [BP76], with more recent approximation and complexity analysis surveyed in [Pas97]. See [CFT99, CFT00] for extensive computational studies of integer programming-based set cover heuristics and exact algorithms. An excellent exposition on algorithms and reduction rules for set cover is presented in [SDK83].

Good expositions of the greedy heuristic for set cover include [CLRS01, Hoc96]. An example demonstrating that the greedy heuristic for set cover can be as bad as $\lg n$ is presented in [Joh74, PS98]. This is not a defect of the heuristic. Indeed, it is provably hard to approximate set cover to within an approximation factor better than $(1 - o(1)) \ln n$ [Fei98].

Related Problems: Matching (see page 498), vertex cover (see page 530), set packing (see page 625).