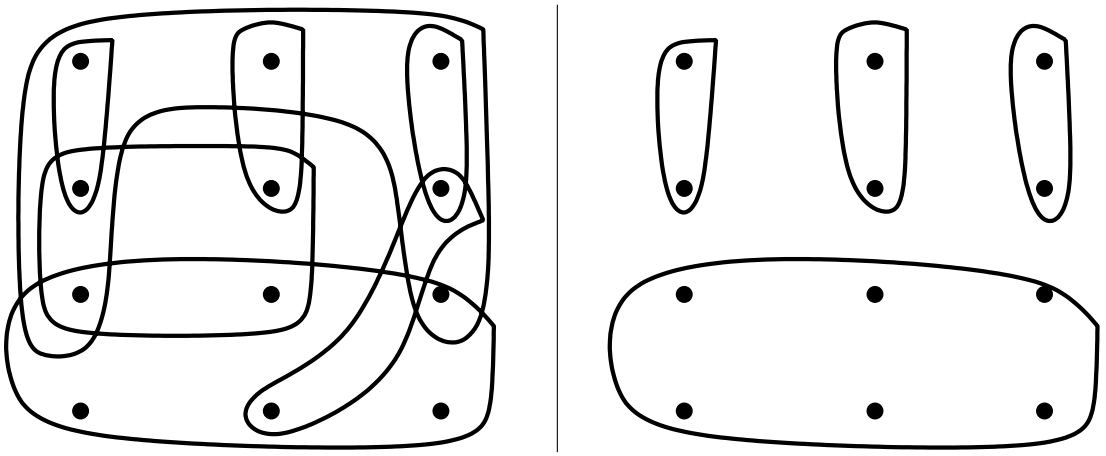

INPUT OUTPUT

18.2 Set Packing

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

Problem description: Select (an ideally small) collection of mutually disjoint subsets from S whose union is the universal set.

Discussion: Set-packing problems arise in applications where we have strong constraints on what is an allowable partition. The key feature of packing problems is that no elements are permitted to be covered by more than one selected subset.

Some flavor of this is captured by the independent set problem in graphs, discussed in Section 16.2 (page 528). There we seek a large subset of vertices from graph G such that each edge is adjacent to at most one of the selected vertices. To model this as set packing, let the universal set consist of all edges of G, and subset S_i consist of all edges incident on vertex v_i. Finally, define an additional singleton set for each edge. Any set packing defines a set of vertices with no edge in common—in other words, an independent set. The singleton sets are used to pick up any edges not covered by the selected vertices.

Scheduling airline flight crews is another application of set packing. Each airplane in the fleet needs to have a crew assigned to it, consisting of a pilot, copilot, and navigator. There are constraints on the composition of possible crews, based on their training to fly different types of aircraft, personality conflicts, and work schedules. Given all possible crew and plane combinations, each represented by a subset of items, we need an assignment such that each plane and each person is

in exactly one chosen combination. After all, the same person cannot be on two different planes simultaneously, and every plane needs a crew. We need a perfect packing given the subset constraints.

Set packing is used here to represent several problems on sets, all of which are NP-complete:

- *Must every element appear in exactly one selected subset?* – In the *exact cover* problem, we seek some collection of subsets such that each element is covered exactly once. The airplane scheduling problem above has the flavor of exact covering, since every plane and crew has to be employed.

  Unfortunately, exact cover puts us in a situation similar to that of Hamiltonian cycle in graphs. If we really *must* cover all the elements exactly once, and this existential problem is NP-complete, then all we can do is exponential search. The cost will be prohibitive unless we happen to stumble upon a solution quickly.

- *Does each element have its own singleton set?* – Things will be far better if we can be content with a partial solution, say by including each element of U as a singleton subset of S. Thus, we can expand any set packing into an exact cover by mopping up the unpacked elements of U with singleton sets. Now our problem is reduced to finding a minimum-cardinality set packing, which can be attacked via heuristics.

- *What is the penalty for covering elements twice?* – In set cover (see Section 18.1 (page 621)), there is no penalty for elements existing in many selected subsets. In exact cover, any such violation is forbidden. For many applications, the truth lies somewhere in between. Such problems can be approached by charging the greedy heuristic more to select a subset that contains previously covered elements.

The right heuristics for set packing are greedy, and similar to those of set cover (see Section 18.1 (page 621)). If we seek a packing with many (few) sets, then we repeatedly select the smallest (largest) subset, delete all subsets from S that clash with it, and repeat. As usual, augmenting this approach with some exhaustive search or randomization (in the form of simulated annealing) is likely to yield better packings at the cost of additional computation.

An alternate and more powerful approach rests on an integer programming formulation akin to that 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

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

to ensure that it is covered by *exactly* one selected subset. Minimizing or maximizing $\sum_i s_i$ while respecting these constraints enables us modulate the desired number of sets in the cover.

Implementations: Since set cover is a more popular and more tractable problem than set packing, it might be easier to find an appropriate implementation to solve the cover problem. Such implementations discussed in Section 18.1 (page 621) should be readily modifiable to support certain packing constraints.

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) for details on FTP-ing these codes.

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

Notes: Survey articles on set packing include [BP76, Pas97]. Bidding strategies for combinatorial auctions typically reduce to solving set-packing problems, as described in [dVV03].

Set-packing relaxations for integer programs are presented in [BW00]. An excellent exposition on algorithms and reduction rules for set packing is presented in [SDK83], including the airplane scheduling application discussed previously.

Related Problems: Independent set (see page 528), set cover (see page 621).