

## Chapter 2

# Integer Programming Models

The importance of integer programming stems from the fact that it can be used to model a vast array of problems arising from the most disparate areas, ranging from practical ones (scheduling, allocation of resources, etc.) to questions in set theory, graph theory, or number theory. We present here a selection of integer programming models, several of which will be further investigated later in this book.

### 2.1 The Knapsack Problem

We are given a knapsack that can carry a maximum weight  $b$  and there are  $n$  types of items that we could take, where an item of type  $i$  has weight  $a_i > 0$ . We want to load the knapsack with items (possibly several items of the same type) without exceeding the knapsack capacity  $b$ . To model this, let a variable  $x_i$  represent the number of items of type  $i$  to be loaded. Then the *knapsack set*

$$S := \left\{ x \in \mathbb{Z}^n : \sum_{i=1}^n a_i x_i \leq b, x \geq 0 \right\}$$

contains precisely all the feasible loads.

If an item of type  $i$  has value  $c_i$ , the problem of loading the knapsack so as to maximize the total value of the load is called the *knapsack problem*. It can be modeled as follows:

$$\max \left\{ \sum_{i=1}^n c_i x_i : x \in S \right\}.$$

If only one unit of each item type can be selected, we use binary variables instead of general integers. The 0, 1 *knapsack set*

$$K := \left\{ x \in \{0, 1\}^n : \sum_{i=1}^n a_i x_i \leq b \right\}$$

can be used to model the 0, 1 *knapsack problem*  $\max\{cx : x \in K\}$ .

## 2.2 Comparing Formulations

Given scalars  $b > 0$  and  $a_j > 0$  for  $j = 1, \dots, n$ , consider the 0, 1 knapsack set  $K := \{x \in \{0, 1\}^n : \sum_{i=1}^n a_i x_i \leq b\}$ . A subset  $C$  of indices is a *cover* for  $K$  if  $\sum_{i \in C} a_i > b$  and it is a *minimal cover* if  $\sum_{i \in C \setminus \{j\}} a_i \leq b$  for every  $j \in C$ . That is,  $C$  is a cover if the knapsack cannot contain all items in  $C$ , and it is minimal if every proper subset of  $C$  can be loaded. Consider the set

$$K^C := \left\{ x \in \{0, 1\}^n : \sum_{i \in C} x_i \leq |C| - 1 \text{ for every minimal cover } C \text{ for } K \right\}.$$

**Proposition 2.1.** *The sets  $K$  and  $K^C$  coincide.*

*Proof.* It suffices to show that (i) if  $C$  is a minimal cover of  $K$ , the inequality  $\sum_{i \in C} x_i \leq |C| - 1$  is valid for  $K$  and (ii) the inequality  $\sum_{i=1}^n a_i x_i \leq b$  is valid for  $K^C$ . The first statement follows from the fact that the knapsack cannot contain all the items in a minimal cover.

Let  $\bar{x}$  be a vector in  $K^C$  and let  $J := \{j : \bar{x}_j = 1\}$ . Suppose  $\sum_{i=1}^n a_i \bar{x}_i > b$  or equivalently  $\sum_{i \in J} a_i > b$ . Let  $C$  be a minimal subset of  $J$  such that  $\sum_{i \in C} a_i > b$ . Then obviously  $C$  is a minimal cover and  $\sum_{i \in C} \bar{x}_i = |C|$ . This contradicts the assumption  $\bar{x} \in K^C$  and the second statement is proved.  $\square$

So the 0, 1 knapsack problem  $\max\{cx : x \in K\}$  can also be formulated as  $\max\{cx : x \in K^C\}$ . The constraints that define  $K$  and  $K^C$  look quite different. The set  $K$  is defined by a single inequality with nonnegative integer coefficients whereas  $K^C$  is defined by many inequalities (their number may be exponential in  $n$ ) whose coefficients are 0, 1. Which of the two formulations is “better”? This question has great computational relevance and

the answer depends on the method used to solve the problem. In this book we focus on algorithms based on linear programming relaxations (remember Sect. 1.2) and for these algorithms, the answer can be stated as follows:

*Assume that  $\{(x, y) : A_1x + G_1y \leq b_1, x \text{ integral}\}$  and  $\{(x, y) : A_2x + G_2y \leq b_2, x \text{ integral}\}$  represent the same mixed integer set  $S$  and consider their linear relaxations  $P_1 = \{(x, y) : A_1x + G_1y \leq b_1\}$ ,  $P_2 = \{(x, y) : A_2x + G_2y \leq b_2\}$ . If  $P_1 \subset P_2$  the first representation is better. If  $P_1 = P_2$  the two representations are equivalent and if  $P_1 \setminus P_2$  and  $P_2 \setminus P_1$  are both nonempty, the two representations are incomparable.*

Next we discuss how to compare the two linear relaxations  $P_1$  and  $P_2$ . If, for every inequality  $a_2x + g_2y \leq \beta_2$  in  $A_2x + G_2y \leq b_2$ , the system

$$uA_1 = a_2, uG_1 = g_2, ub_1 \leq \beta_2, u \geq 0$$

admits a solution  $u \in \mathbb{R}^m$ , where  $m$  is the number of components of  $b_1$ , then every inequality defining  $P_2$  is implied by the set of inequalities that define  $P_1$  and therefore  $P_1 \subseteq P_2$ . Indeed, every point in  $P_1$  satisfies the inequality  $(uA_1)x + (uG_1)y \leq ub_1$  for every nonnegative  $u \in \mathbb{R}^m$ ; so in particular every point in  $P_1$  satisfies  $a_2x + g_2y \leq \beta_2$  whenever  $u$  satisfies the above system.

Farkas's lemma, an important result that will be proved in Chap. 3, implies that the converse is also true if  $P_1$  is nonempty. That is

*Assume  $P_1 \neq \emptyset$ .  $P_1 \subseteq P_2$  if and only if for every inequality  $a_2x + g_2y \leq \beta_2$  in  $A_2x + G_2y \leq b_2$  the system  $uA_1 = a_2$ ,  $uG_1 = g_2$ ,  $ub_1 \leq \beta_2$ ,  $u \geq 0$  is feasible.*

This fact is of fundamental importance in comparing the tightness of different linear relaxations of a mixed integer set. These conditions can be checked by solving linear programs, one for each inequality in  $A_2x + G_2y \leq b_2$ .

We conclude this section with two examples of 0,1 knapsack sets, one where the minimal cover formulation is better than the knapsack formulation and another where the reverse holds. Consider the following 0,1 knapsack set

$$K := \{x \in \{0, 1\}^3 : 3x_1 + 3x_2 + 3x_3 \leq 5\}.$$

Its minimal cover formulation is

$$K^C := \left\{ x \in \{0, 1\}^3 : \begin{array}{rrcr} x_1 & +x_2 & & \leq 1 \\ x_1 & & +x_3 & \leq 1 \\ & x_2 & +x_3 & \leq 1 \end{array} \right\}.$$

The corresponding linear relaxations are the sets

$$P := \{x \in [0, 1]^3 : 3x_1 + 3x_2 + 3x_3 \leq 5\}, \text{ and}$$

$$P^C := \left\{ x \in [0, 1]^3 : \begin{array}{rcl} x_1 & +x_2 & \leq 1 \\ x_1 & & +x_3 \leq 1 \\ & x_2 & +x_3 \leq 1 \end{array} \right\}.$$

respectively. By summing up the three inequalities in  $P^C$  we get

$$2x_1 + 2x_2 + 2x_3 \leq 3$$

which implies  $3x_1 + 3x_2 + 3x_3 \leq 5$ . Thus  $P^C \subseteq P$ . The inclusion is strict since, for instance  $(1, \frac{2}{3}, 0) \in P \setminus P^C$ . In other words, the minimal cover formulation is strictly better than the knapsack formulation in this case.

Now consider a slightly modified example. The knapsack set  $K := \{x \in \{0, 1\}^3 : x_1 + x_2 + x_3 \leq 1\}$  has the same minimal cover formulation  $K^C$  as above, but this time the inclusion is reversed: We have  $P := \{x \in [0, 1]^3 : x_1 + x_2 + x_3 \leq 1\} \subseteq P^C$ . Furthermore  $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}) \in P^C \setminus P$ . In other words, the knapsack formulation is strictly better than the minimal cover formulation in this case.

One can also construct examples where neither formulation is better (Exercise 2.2). In Sect. 7.2.1 we will show how to improve minimal cover inequalities through a procedure called *lifting*.

## 2.3 Cutting Stock: Formulations with Many Variables

A paper mill produces large rolls of paper of width  $W$ , which are then cut into rolls of various smaller widths in order to meet demand. Let  $m$  be the number of different widths that the mill produces. The mill receives an order for  $b_i$  rolls of width  $w_i$  for  $i = 1, \dots, m$ , where  $w_i \leq W$ . How many of the large rolls are needed to meet the order?

To formulate this problem, we may assume that an upper bound  $p$  is known on the number of large rolls to be used. We introduce variables  $j = 1, \dots, n$ , which take value 1 if large roll  $j$  is used and 0 otherwise. Variables  $z_{ij}$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, p$ , indicate the number of small rolls of width  $w_i$  to be cut out of roll  $j$ . Using these variables, one can formulate the cutting stock problem as follows:

$$\begin{aligned}
\min \quad & \sum_{j=1}^p y_j \\
& \sum_{i=1}^m w_i z_{ij} \leq W y_j \quad j = 1, \dots, p \\
& \sum_{j=1}^p z_{ij} \geq b_i \quad i = 1, \dots, m \\
& y_j \in \{0, 1\} \quad j = 1, \dots, p \\
& z_{ij} \in \mathbb{Z}_+ \quad i = 1, \dots, m, j = 1, \dots, p.
\end{aligned} \tag{2.1}$$

The first set of constraints ensures that the sum of the widths of the small rolls cut out of a large roll does not exceed  $W$ , and that a large roll is used whenever a small roll is cut out of it. The second set ensures that the numbers of small rolls that are cut meets the demands. Computational experience shows that this is not a strong formulation: The bound provided by the linear programming relaxation is rather distant from the optimal integer value.

A better formulation is needed. Let us consider all the possible different *cutting patterns*. Each pattern is represented by a vector  $s \in \mathbb{Z}^m$  where component  $i$  represents the number of rolls of width  $w_i$  cut out of the big roll. The set of cutting patterns is therefore  $\mathcal{S} := \{s \in \mathbb{Z}^m : \sum_{i=1}^m w_i s_i \leq W, s \geq 0\}$ . Note that  $\mathcal{S}$  is a knapsack set. For example, if  $W = 5$ , and the order has rolls of 3 different widths  $w_1 = 2.1$ ,  $w_2 = 1.8$  and  $w_3 = 1.5$ , a

possible cutting pattern consists of 3 rolls of width 1.5, i.e.,  $\begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix}$ , another

consists of one roll of width 2.1 and one of width 1.8, i.e.,  $\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$ , etc.

If we introduce integer variables  $x_s$  representing the number of rolls cut according to pattern  $s \in \mathcal{S}$ , the cutting stock problem can be formulated as

$$\begin{aligned}
\min \quad & \sum_{s \in \mathcal{S}} x_s \\
& \sum_{s \in \mathcal{S}} s_i x_s \geq b_i \quad i = 1, \dots, m \\
& x \geq 0 \quad \text{integral.}
\end{aligned} \tag{2.2}$$

This is an integer programming formulation in which the columns of the constraint matrix are all the feasible solutions of a knapsack set. The number of these columns (i.e., the number of possible patterns) is typically enormous, but this is a strong formulation in the sense that the bound provided by the linear programming relaxation is usually close to the optimal value of the integer program. A good solution to the integer program can typically be obtained by simply rounding the linear programming solution. However, solving the linear programming relaxation of (2.2) is challenging. This is best done using column generation, as first proposed by Gilmore and Gomory [168]. We briefly outline this technique here. We will return to it in Sect. 8.2.2. We suggest that readers not familiar with linear programming duality (which will be discussed later in Sect. 3.3) skip directly to Sect. 2.4.

The dual of the linear programming relaxation of (2.2) is:

$$\begin{aligned} \max \quad & \sum_{i=1}^m b_i u_i \\ & \sum_{i=1}^m s_i u_i \leq 1 \quad s \in \mathcal{S} \\ & u \geq 0. \end{aligned} \tag{2.3}$$

Let  $\mathcal{S}'$  be a subset of  $\mathcal{S}$ , and consider the cutting stock problem (2.2) restricted to the variables indexed by  $\mathcal{S}'$ . The dual is the problem defined by the inequalities from (2.3) indexed by  $\mathcal{S}'$ . Let  $\bar{x}$ ,  $\bar{u}$  be optimal solutions to the linear programming relaxations of (2.2) and (2.3) restricted to  $\mathcal{S}'$ . By setting  $\bar{x}_s = 0$ ,  $s \in \mathcal{S} \setminus \mathcal{S}'$ ,  $\bar{x}$  can be extended to a feasible solution of the linear relaxation of (2.2). By strong duality  $\bar{x}$  is an optimal solution of the linear relaxation of (2.2) if  $\bar{u}$  provides a feasible solution to (2.3) (defined over  $\mathcal{S}$ ). The solution  $\bar{u}$  is feasible for (2.3) if and only if  $\sum_{i=1}^m s_i \bar{u}_i \leq 1$  for every  $s \in \mathcal{S}$  or equivalently if and only if the value of the following knapsack problem is at most equal to 1.

$$\max \left\{ \sum_{i=1}^m \bar{u}_i s_i : s \in \mathcal{S} \right\}$$

If the value of this knapsack problem exceeds 1, let  $s^*$  be an optimal solution. Then  $s^*$  corresponds to a constraint of (2.3) that is most violated by  $\bar{u}$ , and  $s^*$  is added to  $\mathcal{S}'$ , thus enlarging the set of candidate patterns.

This is the *column generation* scheme, where variables of a linear program with exponentially many variables are generated on the fly when strong duality is violated, by solving an optimization problem (knapsack, in our case).

## 2.4 Packing, Covering, Partitioning

Let  $E := \{1, \dots, n\}$  be a finite set and  $\mathcal{F} := \{F_1, \dots, F_m\}$  a family of subsets of  $E$ . A set  $S \subseteq E$  is said to be a *packing*, *partitioning* or *covering* of the family  $\mathcal{F}$  if  $S$  intersects each member of  $\mathcal{F}$  *at most once*, *exactly once*, or *at least once*, respectively. Representing a set  $S \subseteq E$  by its *characteristic vector*  $x^S \in \{0, 1\}^n$ , i.e.,  $x_j^S = 1$  if  $j \in S$ , and  $x_j^S = 0$  otherwise, the families of packing, partitioning and covering sets have the following formulations.

$$\begin{aligned} S^P &:= \{x \in \{0, 1\}^n : \sum_{j \in F_i} x_j \leq 1, \forall F_i \in \mathcal{F}\}, \\ S^T &:= \{x \in \{0, 1\}^n : \sum_{j \in F_i} x_j = 1, \forall F_i \in \mathcal{F}\}, \\ S^C &:= \{x \in \{0, 1\}^n : \sum_{j \in F_i} x_j \geq 1, \forall F_i \in \mathcal{F}\}. \end{aligned}$$

Given weights  $w_j$  on the elements  $j = 1, \dots, n$ , the *set packing* problem is  $\max\{\sum_{j=1}^n w_j x_j : x \in S^P\}$ , the *set partitioning* problem is  $\min\{\sum_{j=1}^n w_j x_j : x \in S^T\}$ , and the *set covering* problem is  $\min\{\sum_{j=1}^n w_j x_j : x \in S^C\}$ .

Given  $E := \{1, \dots, n\}$  and a family  $\mathcal{F} := \{F_1, \dots, F_m\}$  of subsets of  $E$ , the *incidence matrix* of  $\mathcal{F}$  is the  $m \times n$  0, 1 matrix in which  $a_{ij} = 1$  if and only if  $j \in F_i$ . Then  $S^P = \{x \in \{0, 1\}^n : Ax \leq \mathbf{1}\}$ , where  $\mathbf{1}$  denotes the column vector in  $\mathbb{R}^m$  all of whose components are equal to 1. Similarly the sets  $S^T$ ,  $S^C$  can be expressed in terms of  $A$ . Conversely, given any 0,1 matrix  $A$ , one can define a *set packing family*  $S^P(A) := \{x \in \{0, 1\}^n : Ax \leq \mathbf{1}\}$ . The families  $S^T(A)$ ,  $S^C(A)$  are defined similarly.

Numerous practical problems and several problems in combinatorics and graph theory can be formulated as set packing or covering. We illustrate some of them.

### 2.4.1 Set Packing and Stable Sets

Let  $G = (V, E)$  be an undirected graph and let  $n := |V|$ . A *stable set* in  $G$  is a set of nodes no two of which are adjacent. Therefore  $S \subseteq V$  is a stable set if and only if its characteristic vector  $x \in \{0, 1\}^n$  satisfies  $x_i + x_j \leq 1$  for every edge  $ij \in E$ . If we consider  $E$  as a family of subsets of  $V$ , the characteristic vectors of the stable sets in  $G$  form a set packing family, namely

$$\text{stab}(G) := \{x \in \{0, 1\}^n : x_i + x_j \leq 1 \text{ for all } ij \in E\}.$$

We now show the converse: Every set packing family is the family of characteristic vectors of the stable sets of some graph. Given an  $m \times n$  0,1 matrix  $A$ , the *intersection graph* of  $A$  is an undirected simple graph

$G_A = (V, E)$  on  $n$  nodes, corresponding to the columns of  $A$ . Two nodes  $u, v$  are adjacent in  $G_A$  if and only if  $a_{iu} = a_{iv} = 1$  for some row index  $i$ ,  $1 \leq i \leq m$ . In Fig. 2.1 we show a matrix  $A$  and its intersection graph.

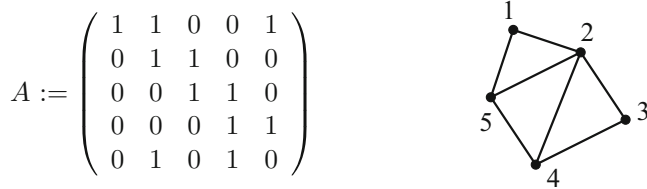


Figure 2.1: A 0,1 matrix  $A$  and its intersection graph  $G_A$

We have  $S^P(A) = \text{stab}(G_A)$  since a vector  $x \in \{0,1\}^n$  is in  $S^P(A)$  if and only if  $x_j + x_k \leq 1$  whenever  $a_{ij} = a_{ik} = 1$  for some row  $i$ .

All modern integer programming solvers use intersection graphs to model logical conditions among the binary variables of integer programming formulations. Nodes are often introduced for the complement of binary variables as well: This is useful to model conditions such as  $x_i \leq x_j$ , which can be reformulated in set packing form as  $x_i + (1 - x_j) \leq 1$ . In this context, the intersection graph is called a *conflict graph*. We refer to the paper of Atamtürk, Nemhauser, and Savelsbergh [16] for the use of conflict graphs in integer programming. This paper stresses the practical importance of strengthening set packing formulations.

### 2.4.2 Strengthening Set Packing Formulations

Given a 0,1 matrix  $A$ , the correspondence between  $S^P(A)$  and  $\text{stab}(G_A)$  can be used to strengthen the formulation  $\{x \in \{0,1\}^n : Ax \leq \mathbf{1}\}$ .

A *clique* in a graph is a set of pairwise adjacent nodes. Since a clique  $K$  in  $G_A$  intersects any stable set in at most one node, the inequality

$$\sum_{j \in K} x_j \leq 1$$

is valid for  $S^P(A) = \text{stab}(G_A)$ . This inequality is called a *clique inequality*. Conversely, given  $Q \subseteq V$ , if  $\sum_{j \in Q} x_j \leq 1$  is a valid inequality for  $\text{stab}(G_A)$ , then every pair of nodes in  $Q$  must be adjacent, that is,  $Q$  is a clique of  $G_A$ .

A clique is *maximal* if it is not properly contained in any other clique.

Note that, given two cliques  $K, K'$  in  $G_A$  such that  $K \subset K'$ , inequality  $\sum_{j \in K} x_j \leq 1$  is implied by the inequalities  $\sum_{j \in K'} x_j \leq 1$  and  $x_j \geq 0$ ,  $j \in K' \setminus K$ .



On the other hand, the following argument shows that no maximal clique inequality is implied by the other clique inequalities and the constraints  $0 \leq x_j \leq 1$ . Let  $K$  be a maximal clique of  $G_A$ . We will exhibit a point  $\bar{x} \in [0, 1]^V$  that satisfies all clique inequalities except for the one relative to  $K$ . Let  $\bar{x}_j := \frac{1}{|K|-1}$  for all  $j \in K$  and  $\bar{x}_j := 0$  otherwise. Since  $K$  is a maximal clique, every other clique  $K'$  intersects it in at most  $|K| - 1$  nodes, therefore  $\sum_{j \in K'} \bar{x}_j \leq 1$ . On the other hand,  $\sum_{j \in K} \bar{x}_j = 1 + \frac{1}{|K|-1} > 1$ . We have shown the following.

**Theorem 2.2.** *Given an  $m \times n$  0,1 matrix  $A$ , let  $\mathcal{K}$  be the collection of all maximal cliques of its intersection graph  $G_A$ . The strongest formulation for  $SP(A) = \text{stab}(G_A)$  that only involves set packing constraints is*

$$\{x \in \{0, 1\}^n : \sum_{j \in K} x_j \leq 1, \forall K \in \mathcal{K}\}.$$

**Example 2.3.** In the example of Fig. 2.1, the inequalities  $x_2 + x_3 + x_4 \leq 1$  and  $x_2 + x_4 + x_5 \leq 1$  are clique inequalities relative to the cliques  $\{2, 3, 4\}$  and  $\{2, 4, 5\}$  in  $G_A$ . Note that the point  $(0, 1/2, 1/2, 1/2, 0)$  satisfies  $Ax \leq 1$ ,  $0 \leq x \leq 1$  but violates  $x_2 + x_3 + x_4 \leq 1$ . A better formulation of  $Ax \leq 1$ ,  $x \in \{0, 1\}^n$  is obtained by replacing the constraint matrix  $A$  by the maximal clique versus node incidence matrix  $A_c$  of the intersection graph of  $A$ . For

the example of Fig. 2.1,  $A_c := \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$ . The reader can verify

that this formulation is perfect, as defined in Sect. 1.4. ■

Note that the strongest set packing formulation described in Theorem 2.2 may contain exponentially many inequalities. If  $\mathcal{K}$  denotes the collection of all maximal cliques of a graph  $G$ , the  $|\mathcal{K}| \times n$  incidence matrix of  $\mathcal{K}$  is called the *clique matrix* of  $G$ . Exercise 2.10 gives a characterization of clique matrices due to Gilmore [167]. Theorem 2.2 prompts the following question:

*For which graphs  $G$  is the formulation defined in Theorem 2.2 a perfect formulation of  $\text{stab}(G)$ ?*

This leads to the theory of perfect graphs, see Sect. 4.11 for references on this topic. In Chap. 10 we will discuss a semidefinite relaxation of  $\text{stab}(G)$ .

### 2.4.3 Set Covering and Transversals

We have seen the equivalence between general packing sets and stable sets in graphs. Covering sets do not seem to have an equivalent graphical representation. However some important questions in graph theory regarding

connectivity, coloring, parity, and others can be formulated using covering sets. We first introduce a general setting for these formulations.

Given a finite set  $E := \{1, \dots, n\}$ , a family  $\mathcal{S} := \{S_1, \dots, S_m\}$  of subsets  $E$  is a *clutter* if it has the following property:

*For every pair  $S_i, S_j \in \mathcal{S}$ , both  $S_i \setminus S_j$  and  $S_j \setminus S_i$  are nonempty.*

A subset  $T$  of  $E$  is a *transversal* of  $\mathcal{S}$  if  $T \cap S_i \neq \emptyset$  for every  $S_i \in \mathcal{S}$ . Let  $\mathcal{T} := \{T_1, \dots, T_q\}$  be the family of all inclusionwise minimal transversals of  $\mathcal{S}$ . The family  $\mathcal{T}$  is a clutter as well, called the *blocker* of  $\mathcal{S}$ . The following set-theoretic property, due to Lawler [251] and Edmonds and Fulkerson [127], is fundamental for set covering formulations.

**Proposition 2.4.** *Let  $\mathcal{S}$  be a clutter and  $\mathcal{T}$  its blocker. Then  $\mathcal{S}$  is the blocker of  $\mathcal{T}$ .*

*Proof.* Let  $\mathcal{Q}$  be the blocker of  $\mathcal{T}$ . We need to show that  $\mathcal{Q} = \mathcal{S}$ . By definition of clutter, it suffices to show that every member of  $\mathcal{S}$  contains some member of  $\mathcal{Q}$  and every member of  $\mathcal{Q}$  contains some member of  $\mathcal{S}$ .

Let  $S_i \in \mathcal{S}$ . By definition of  $\mathcal{T}$ ,  $S_i \cap T \neq \emptyset$  for every  $T \in \mathcal{T}$ . Therefore  $S_i$  is a transversal of  $\mathcal{T}$ . Because  $\mathcal{Q}$  is the blocker of  $\mathcal{T}$ , this implies that  $S_i$  contains a member of  $\mathcal{Q}$ .

We now show the converse, namely every member of  $\mathcal{Q}$  contains a member of  $\mathcal{S}$ . Suppose not. Then there exists a member  $Q$  of  $\mathcal{Q}$  such that  $(E \setminus Q) \cap S \neq \emptyset$  for every  $S \in \mathcal{S}$ . Therefore  $E \setminus Q$  is a transversal of  $\mathcal{S}$ . This implies that  $E \setminus Q$  contains some member  $T \in \mathcal{T}$ . But then  $Q \cap T = \emptyset$ , a contradiction to the assumption that  $Q$  is a transversal of  $\mathcal{T}$ .  $\square$

In light of the previous proposition, we call the pair of clutters  $\mathcal{S}$  and its blocker  $\mathcal{T}$  a *blocking pair*.

Given a vector  $x \in \mathbb{R}^n$ , the *support* of  $x$  is the set  $\{i \in \{1, \dots, n\} : x_i \neq 0\}$ . Proposition 2.4 yields the following:

**Observation 2.5.** *Let  $\mathcal{S}, \mathcal{T}$  be a blocking pair of clutters and let  $A$  be the incidence matrix of  $\mathcal{T}$ . The vectors with minimal support in the set covering family  $S^C(A)$  are the characteristic vectors of the family  $\mathcal{S}$ .*

Consider the following problem, which arises often in combinatorial optimization (we give three examples in Sect. 2.4.4).

*Let  $E := \{1, \dots, n\}$  be a set of elements where each element  $j = 1, \dots, n$  is assigned a nonnegative weight  $w_j$ , and let  $\mathcal{R}$  be a family of subsets of  $E$ . Find a member  $S \in \mathcal{R}$  having minimum weight  $\sum_{j \in S} w_j$ .*

Let  $\mathcal{S}$  be the clutter consisting of the minimal members of  $\mathcal{R}$ . Note that, since the weights are nonnegative, the above problem always admit an optimal solution that is a member of  $\mathcal{S}$ .

Let  $\mathcal{T}$  be the blocker of  $\mathcal{S}$  and  $A$  the incidence matrix of  $\mathcal{T}$ . In light of Observation 2.5 an integer programming formulation for the above problem is given by

$$\min\{wx : x \in S^C(A)\}.$$

#### 2.4.4 Set Covering on Graphs: Many Constraints

We now apply the technique introduced above to formulate some optimization problems on an undirected graph  $G = (V, E)$  with nonnegative edge weights  $w_e$ ,  $e \in E$ .

Given  $S \subseteq V$ , let  $\delta(S) := \{uv \in E : u \in S, v \notin S\}$ . A *cut* in  $G$  is a set  $F$  of edges such that  $F = \delta(S)$  for some  $S \subseteq V$ . A cut  $F$  is *proper* if  $F = \delta(S)$  for some  $\emptyset \neq S \subset V$ . For every node  $v$ , we will write  $\delta(v) := \delta(\{v\})$  to denote the set of edges containing  $v$ . The *degree* of node  $v$  is  $|\delta(v)|$ .

##### Minimum Weight $s, t$ -Cuts

Let  $s, t$  be two distinct nodes of a connected graph  $G$ . An  $s, t$ -cut is a cut of the form  $\delta(S)$  such that  $s \in S$  and  $t \notin S$ . Given nonnegative weights on the edges,  $w_e$  for  $e \in E$ , the *minimum weight  $s, t$ -cut problem* is to find an  $s, t$ -cut  $F$  that minimizes  $\sum_{e \in F} w_e$ .

An  $s, t$ -path in  $G$  is a path between  $s$  and  $t$  in  $G$ . Equivalently, an  $s, t$ -path is a minimal set of edges that induce a connected graph containing both  $s$  and  $t$ . Let  $\mathcal{S}$  be the family of inclusionwise minimal  $s, t$ -cuts. Note that its blocker  $\mathcal{T}$  is the family of  $s, t$ -paths. Therefore the minimum weight  $s, t$ -cut problem can be formulated as follows:

$$\begin{array}{ll} \min & \sum_{e \in E} w_e x_e \\ & \sum_{e \in P} x_e \geq 1 \quad \text{for all } s, t\text{-paths } P \\ & x_e \in \{0, 1\} \quad e \in E. \end{array}$$

Fulkerson [156] showed that the above formulation is a perfect formulation. Ford and Fulkerson [146] gave a polynomial-time algorithm for the minimum weight of an  $s, t$ -cut problem, and proved that the minimum weight of an  $s, t$ -cut is equal to the maximum value of an  $s, t$ -flow. This will be discussed in Chap. 4.

Let  $A$  be the incidence matrix of a clutter and  $B$  the incidence matrix of its blocker. Lehman [254] proved that  $S^C(A)$  is a perfect formulation if

and only if  $S^C(B)$  is a perfect formulation. Lehman's theorem together with Fulkerson's theorem above imply that the following linear program solves the shortest  $s, t$ -path problem when  $w \geq 0$ :

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e \\ & \sum_{e \in C} x_e \geq 1 \quad \text{for all } s, t\text{-cuts } C \\ & 0 \leq x_e \leq 1 \quad e \in E. \end{aligned}$$

We give a more traditional formulation of the shortest  $s, t$ -path problem in Sect. 4.3.2.

### Minimum Cut

In the *min-cut problem* one wants to find a proper cut of minimum total weight in a connected graph  $G$  with nonnegative edge weights  $w_e$ ,  $e \in E$ . An edge set  $T \subseteq E$  is a *spanning tree* of  $G$  if it is an inclusionwise minimal set of edges such that the graph  $(V, T)$  is connected.

Let  $\mathcal{S}$  be the family of inclusionwise minimal proper cuts. Note that the blocker of  $\mathcal{S}$  is the family of spanning trees of  $G$ , hence one can formulate the min-cut problem as

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e \\ & \sum_{e \in T} x_e \geq 1 \quad \text{for all spanning trees } T \\ & x_e \in \{0, 1\} \quad e \in E. \end{aligned} \tag{2.4}$$

This is not a perfect formulation (see Exercise 2.13). Nonetheless, the min-cut problem is polynomial-time solvable. A solution can be found by fixing a node  $s \in V$ , computing a minimum weight  $s, t$ -cut for every choice of  $t$  in  $V \setminus \{s\}$ , and selecting the cut of minimum weight among the  $|V| - 1$  cuts computed.

### Max-Cut

Given a graph  $G = (V, E)$  with edge weights  $w_e$ ,  $e \in E$ , the *max-cut problem* asks to find a set  $F \subseteq E$  of maximum total weight in  $G$  such that  $F$  is a cut of  $G$ . That is,  $F = \delta(S)$ , for some  $S \subseteq V$ .

Given a graph  $G = (V, E)$  and  $C \subseteq E$ , let  $V(C)$  denote the set of nodes that belong to at least one edge in  $C$ . A set of edges  $C \subseteq E$  is a *cycle* in  $G$  if the graph  $(V(C), C)$  is connected and all its nodes have degree two. A cycle  $C$  in  $G$  is an *odd cycle* if  $C$  has an odd number of edges. A basic fact in graph theory states that a set  $F \subseteq E$  is contained in a cut of  $G$  if and only if  $(E \setminus F) \cap C \neq \emptyset$  for every odd cycle  $C$  of  $G$  (see Exercise 2.14).

Therefore when  $w_e \geq 0$ ,  $e \in E$ , the max-cut problem in the graph  $G = (V, E)$  can be formulated as the problem of finding a set  $E' \subseteq E$  of minimum total weight such that  $E' \cap C \neq \emptyset$  for every odd cycle  $C$  of  $G$ .

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e \\ & \sum_{e \in C} x_e \geq 1 \quad \text{for all odd cycles } C \\ & x_e \in \{0, 1\} \quad e \in E. \end{aligned} \tag{2.5}$$

Given an optimal solution  $\bar{x}$  to (2.5), the optimal solution of the max-cut problem is the cut  $\{e \in E : \bar{x}_e = 0\}$ .

Unlike the two previous examples, the max-cut problem is NP-hard. However, Goemans and Williamson [173] show that a near-optimal solution can be found in polynomial time, using a semidefinite relaxation that will be discussed in Sect. 10.2.1.

### 2.4.5 Set Covering with Many Variables: Crew Scheduling

An airline wants to operate its daily flight schedule using the smallest number of crews. A crew is on duty for a certain number of consecutive hours and therefore may operate several flights. A feasible crew schedule is a sequence of flights that may be operated by the same crew within its duty time. For instance it may consist of the 8:30–10:00 am flight from Pittsburgh to Chicago, then the 11:30 am–1:30 pm Chicago–Atlanta flight and finally the 2:45–4:30 pm Atlanta–Pittsburgh flight.

Define  $A = \{a_{ij}\}$  to be the 0,1 matrix whose rows correspond to the daily flights operated by the company and whose columns correspond to all the possible crew schedules. The entry  $a_{ij}$  equals 1 if flight  $i$  is covered by crew schedule  $j$ , and 0 otherwise. The problem of minimizing the number of crews can be formulated as

$$\min \left\{ \sum_j x_j : x \in S^C(A) \right\}.$$

In an optimal solution a flight may be covered by more than one crew: One crew operates the flight and the other occupies passenger seats. This is why the above formulation involves covering constraints. The number of columns (that is, the number of possible crew schedules) is typically enormous. Therefore, as in the cutting stock example, column generation is relevant in crew scheduling applications.

### 2.4.6 Covering Steiner Triples

Fulkerson, Nemhauser, and Trotter [157] constructed set covering problems of small size that are notoriously difficult to solve. A *Steiner triple system* of order  $n$  (denoted by  $STS(n)$ ) consists of a set  $E$  of  $n$  elements and a collection  $\mathcal{S}$  of triples of  $E$  with the property that every pair of elements in  $E$  appears together in a unique triple in  $\mathcal{S}$ . It is known that a Steiner triple system of order  $n$  exists if and only if  $n \equiv 1$  or  $3 \pmod{6}$ . A subset  $C$  of  $E$  is a covering of the Steiner triple system if  $C \cap T \neq \emptyset$  for every triple  $T$  in  $\mathcal{S}$ . Given a Steiner triple system, the problem of computing the smallest cardinality of a cover is

$$\min\left\{\sum_j x_j : x \in S^C(A)\right\}$$

where  $A$  is the  $|\mathcal{S}| \times n$  incidence matrix of the collection  $\mathcal{S}$ . Fulkerson, Nemhauser, and Trotter constructed an infinite family of Steiner triple systems in 1974 and asked for the smallest cardinality of a cover. The question was solved 5 years later for  $STS(45)$ , it took another 15 years for  $STS(81)$ , and the current record is the solution of  $STS(135)$  and  $STS(243)$  [292].

## 2.5 Generalized Set Covering: The Satisfiability Problem

We generalize the set covering model by allowing constraint matrices whose entries are  $0, \pm 1$  and we use it to formulate problems in propositional logic.

*Atomic propositions*  $x_1, \dots, x_n$  can be either *true* or *false*. A *truth assignment* is an assignment of “true” or “false” to every atomic proposition. A *literal*  $L$  is an atomic proposition  $x_j$  or its negation  $\neg x_j$ . A *conjunction* of two literals  $L_1 \wedge L_2$  is true if both literals are true and a *disjunction* of two literals  $L_1 \vee L_2$  is true if at least one of  $L_1, L_2$  is true. A *clause* is a disjunction of literals and it is *satisfied* by a given truth assignment if at least one of its literals is true.

For example, the clause with three literals  $x_1 \vee x_2 \vee \neg x_3$  is satisfied if “ $x_1$  is true or  $x_2$  is true or  $x_3$  is false.” In particular, it is satisfied by the truth assignment  $x_1 = x_2 = x_3 = \text{“false.”}$

It is usual to identify truth assignments with 0,1 vectors:  $x_i = 1$  if  $x_i = \text{"true"}$  and  $x_i = 0$  if  $x_i = \text{"false"}$ . A truth assignment satisfies the clause

$$\bigvee_{j \in P} x_j \vee \left( \bigvee_{j \in N} \neg x_j \right)$$

if and only if the corresponding 0,1 vector satisfies the inequality

$$\sum_{j \in P} x_j - \sum_{j \in N} x_j \geq 1 - |N|.$$

For example the clause  $x_1 \vee x_2 \vee \neg x_3$  is satisfied if and only if the corresponding 0,1 vector satisfies the inequality  $x_1 + x_2 - x_3 \geq 0$ .

A logic statement consisting of a conjunction of clauses is said to be in *conjunctive normal form*. For example the logical proposition  $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3)$  is in conjunctive normal form. Such logic statements can be represented by a system of  $m$  linear inequalities, where  $m$  is the number of clauses in the conjunctive normal form. This can be written in the form:

$$Ax \geq \mathbf{1} - n(A) \tag{2.6}$$

where  $A$  is an  $m \times n$   $0, \pm 1$  matrix and the  $i^{\text{th}}$  component of  $n(A)$  is the number of  $-1$ 's in row  $i$  of  $A$ . For example the logical proposition  $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3)$  corresponds to the system of constraints

$$\begin{aligned} x_1 + x_2 - x_3 &\geq 0 \\ x_2 + x_3 &\geq 1 \\ x_i &\in \{0, 1\}^3. \end{aligned}$$

In this example  $A = \begin{pmatrix} 1 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix}$  and  $n(A) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ .

Every logic statement can be written in conjunctive normal form by using rules of logic such as  $L_1 \vee (L_2 \wedge L_3) = (L_1 \vee L_2) \wedge (L_1 \vee L_3)$ ,  $\neg(L_1 \wedge L_2) = \neg L_1 \vee \neg L_2$ , etc. This will be illustrated in Exercises 2.24, 2.25.

We present two classical problems in logic.

The *satisfiability problem* (SAT) for a set  $S$  of clauses, asks for a truth assignment satisfying all the clauses in  $S$  or a proof that none exists. Equivalently, SAT consists of finding a 0,1 solution  $x$  to (2.6) or showing that none exists.

*Logical inference* in propositional logic consists of a set  $S$  of clauses (the premises) and a clause  $C$  (the conclusion), and asks whether every truth

assignment satisfying all the clauses in  $S$  also satisfies the conclusion  $C$ . To the clause  $C$ , we associate the inequality

$$\sum_{j \in P(C)} x_j - \sum_{j \in N(C)} x_j \geq 1 - |N(C)|. \quad (2.7)$$

Therefore the conclusion  $C$  cannot be deduced from the premises  $S$  if and only if (2.6) has a 0,1 solution that violates (2.7).

Equivalently  $C$  cannot be deduced from  $S$  if and only if the integer program

$$\min \left\{ \sum_{j \in P(C)} x_j - \sum_{j \in N(C)} x_j : Ax \geq \mathbf{1} - n(A), x \in \{0,1\}^n \right\}$$

has a solution with value  $-|n(C)|$ .

## 2.6 The Sudoku Game

The game is played on a  $9 \times 9$  grid which is subdivided into 9 blocks of  $3 \times 3$  contiguous cells. The grid must be filled with numbers  $1, \dots, 9$  so that all the numbers between 1 and 9 appear in each row, in each column and in each of the nine blocks. A game consists of an initial assignment of numbers in some cells (Fig. 2.2).

8				2	6			
						7		4
			7					5
			1				3	6
	1			8			4	
9	8				3			
3					1			
7		5						
			2	5				8

Figure 2.2: An instance of the Sudoku game

This is a decision problem that can be modeled with binary variables  $x_{ijk}$ ,  $1 \leq i, j, k \leq 9$  where  $x_{ijk} = 1$  if number  $k$  is entered in position with coordinates  $i, j$  of the grid, and 0 otherwise.



The constraints are:

$$\begin{aligned}
 \sum_{j=1}^9 x_{ijk} &= 1, & 1 \leq j, k \leq 9 & \quad (\text{each number } k \text{ appears once in column } j) \\
 \sum_{j=1}^9 x_{ijk} &= 1, & 1 \leq i, k \leq 9 & \quad (\text{each } k \text{ appears once in row } i) \\
 \sum_{q,r=0}^2 x_{i+q,j+r,k} &= 1, & i, j = 1, 4, 7, 1 \leq k \leq 9 & \quad (\text{each } k \text{ appears once in a block}) \\
 \sum_{k=1}^9 x_{ijk} &= 1, & 1 \leq i, j \leq 9 & \quad (\text{each cell contains exactly one number}) \\
 x_{ijk} &\in \{0, 1\}, & 1 \leq i, j, k \leq 9 & \\
 x_{ijk} &= 1, & \text{when the initial assignment has number } k \text{ in cell } i, j. &
 \end{aligned}$$

In *constraint programming*, variables take values in a specified domain, which may include data that are non-quantitative, and constraints restrict the space of possibilities in a way that is more general than the one given by linear constraints. We refer to the book “Constraint Processing” by R. Dechter [108] for an introduction to constraint programming. One of these constraints is `\alldifferent` $\{z_1, \dots, z_n\}$  which forces variables  $z_1, \dots, z_n$  to take distinct values in the domain. Using the `\alldifferent` $\{\}$  constraint, we can formulate the Sudoku game using 2-index variables, instead of the 3-index variables used in the above integer programming formulation. Variable  $x_{ij}$  represents the value in the cell of the grid with coordinates  $(i, j)$ . Thus  $x_{ij}$  take its values in the domain  $\{1, \dots, 9\}$  and there is an `\alldifferent` $\{\}$  constraint that involves the set of variables in each row, each column and each of the nine blocks.

## 2.7 The Traveling Salesman Problem

This section illustrates the fact that several formulations may exist for a given problem, and it is not immediately obvious which is the best for branch-and-cut algorithms.

A traveling salesman must visit  $n$  cities and return to the city he started from. We will call this a *tour*. Given the cost  $c_{ij}$  of traveling from city  $i$  to city  $j$ , for each  $1 \leq i, j \leq n$  with  $i \neq j$ , in which order should the salesman visit the cities to minimize the total cost of his tour? This problem is the famous *traveling salesman problem*. If we allow costs  $c_{ij}$  and  $c_{ji}$  to be different for any given pair of cities  $i, j$ , then the problem is referred to as the *asymmetric traveling salesman problem*, while if  $c_{ij} = c_{ji}$  for every pair of cities  $i$  and  $j$ , the problem is known as the *symmetric traveling salesman problem*. In Fig. 2.3, the left diagram represents eight cities in the plane. The cost of traveling between any two cities is assumed to be proportional to the Euclidean distance between them. The right diagram depicts the optimal tour.



Figure 2.3: An instance of the symmetric traveling salesman problem in the Euclidean plane, and the optimal tour

It will be convenient to define the traveling salesman problem on a graph (directed or undirected). Given a *digraph* (a directed graph)  $D = (V, A)$ , a (*directed*) *Hamiltonian tour* is a circuit that traverses each node exactly once. Given costs  $c_a$ ,  $a \in A$ , the asymmetric traveling salesman problem on  $D$  consists in finding a Hamiltonian tour in  $D$  of minimum total cost. Note that, in general,  $D$  might not contain any Hamiltonian tour. We give three different formulations for the asymmetric traveling salesman problem.

The first formulation is due to Dantzig, Fulkerson, and Johnson [103]. They introduce a binary variable  $x_{ij}$  for all  $ij \in A$ , where  $x_{ij} = 1$  if the tour visits city  $j$  immediately after city  $i$ , and 0 otherwise. Given a set of cities  $S \subseteq V$ , let  $\delta^+(S) := \{ij \in A : i \in S, j \notin S\}$ , and let  $\delta^-(S) := \{ij \in A : i \notin S, j \in S\}$ . For ease of notation, for  $v \in V$  we use  $\delta^+(v)$  and  $\delta^-(v)$  instead of  $\delta^+(\{v\})$  and  $\delta^-(\{v\})$ . The Dantzig–Fulkerson–Johnson formulation of the traveling salesman problem is as follows.

$$\min \sum_{a \in A} c_a x_a \quad (2.8)$$

$$\sum_{a \in \delta^+(i)} x_a = 1 \quad \text{for } i \in V \quad (2.9)$$

$$\sum_{a \in \delta^-(i)} x_a = 1 \quad \text{for } i \in V \quad (2.10)$$

$$\sum_{a \in \delta^+(S)} x_a \geq 1 \quad \text{for } \emptyset \subset S \subset V \quad (2.11)$$

$$x_a \in \{0, 1\} \quad \text{for } a \in A. \quad (2.12)$$

Constraints (2.9)–(2.10), known as *degree constraints*, guarantee that the tour visits each node exactly once and constraints (2.11) guarantee that the solution does not decompose into several subtours. Constraints (2.11) are known under the name of *subtour elimination constraints*. Despite the

exponential number of constraints, this is the formulation that is most widely used in practice. Initially, one solves the linear programming relaxation that only contains (2.9)–(2.10) and  $0 \leq x_{ij} \leq 1$ . The subtour elimination constraints are added later, on the fly, only when needed. This is possible because the so-called separation problem can be solved efficiently for such constraints (see Chap. 4).

Miller, Tucker and Zemlin [278] found a way to avoid the subtour elimination constraints (2.11). Assume  $V = \{1, \dots, n\}$ . The formulation has extra variables  $u_i$  that represent the position of node  $i \geq 2$  in the tour, assuming that the tour starts at node 1, i.e., node 1 has position 1. Their formulation is identical to (2.8)–(2.12) except that (2.11) is replaced by

$$u_i - u_j + 1 \leq n(1 - x_{ij}) \quad \text{for all } ij \in A, i, j \neq 1. \quad (2.13)$$

It is not difficult to verify that the Miller–Tucker–Zemlin formulation is correct. Indeed, if  $x$  is the incident vector of a tour, define  $u_i$  to be the position of node  $i$  in the tour, for  $i \geq 2$ . Then constraint (2.13) is satisfied. Conversely, if  $x \in \{0, 1\}^E$  satisfies (2.9)–(2.10) but is not the incidence vector of a tour, then (2.9)–(2.10) and (2.12) imply that there is at least one subtour  $C \subseteq A$  that does not contain node 1. Summing the inequalities (2.13) relative to every  $ij \in C$  gives the inequality  $|C| \leq 0$ , a contradiction. Therefore, if (2.9)–(2.10), (2.12), (2.13) are satisfied,  $x$  must represent a tour. Although the Miller–Tucker–Zemlin formulation is correct, we will show in Chap. 4 that it produces weaker bounds for branch-and-cut algorithms than the Dantzig–Fulkerson–Johnson formulation. It is for this reason that the latter is preferred in practice.

It is also possible to formulate the traveling salesman problem using variables  $x_{ak}$  for every  $a \in A$ ,  $k \in V$ , where  $x_{ak} = 1$  if arc  $a$  is the  $k$ th leg of the Hamiltonian tour, and  $x_{ak} = 0$  otherwise. The traveling salesman problem can be formulated as follows.

$$\begin{aligned} \min \quad & \sum_{a \in A} \sum_k c_a x_{ak} \\ & \sum_{a \in \delta^+(i)} \sum_k x_{ak} = 1 \quad \text{for } i = 1, \dots, n \\ & \sum_{a \in \delta^-(i)} \sum_k x_{ak} = 1 \quad \text{for } i = 1, \dots, n \end{aligned} \quad (2.14)$$

$$\begin{aligned}
\sum_{a \in A} x_{ak} &= 1 \quad \text{for } k = 1, \dots, n \\
\sum_{a \in \delta^-(i)} x_{ak} &= \sum_{a \in \delta^+(i)} x_{a,k+1} \quad \text{for } i = 1, \dots, n \text{ and } k = 1, \dots, n-1 \\
\sum_{a \in \delta^-(1)} x_{an} &= \sum_{a \in \delta^+(1)} x_{a1} = 1 \\
x_{ak} &= 0 \text{ or } 1 \quad \text{for } a \in A, k = 1, \dots, n.
\end{aligned}$$

The first three constraints impose that each city is entered once, left once, and each leg of the tour contains a unique arc. The next constraint imposes that if leg  $k$  brings the salesman to city  $i$ , then he leaves city  $i$  on leg  $k+1$ . The last constraint imposes that the first leg starts from city 1 and the last returns to city 1. The main drawback of this formulation is its large number of variables.

The Dantzig–Fulkerson–Johnson formulation has a simple form in the case of the symmetric traveling salesman problem. Given an undirected graph  $G = (V, E)$ , a *Hamiltonian tour* is a cycle that goes exactly once through each node of  $G$ . Given costs  $c_e$ ,  $e \in E$ , the symmetric traveling salesman problem is to find a Hamiltonian tour in  $G$  of minimum total cost. The Dantzig–Fulkerson–Johnson formulation for the symmetric traveling salesman problem is the following.

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e \\
& \sum_{e \in \delta(i)} x_e = 2 \quad \text{for } i \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 \quad \text{for } \emptyset \subset S \subset V \\
& x_e \in \{0, 1\} \quad \text{for } e \in E.
\end{aligned} \tag{2.15}$$

In this context  $\sum_{e \in \delta(i)} x_e = 2$  for  $i \in V$  are the *degree constraints* and  $\sum_{e \in \delta(S)} x_e \geq 2$  for  $\emptyset \subset S \subset V$  are the *subtour elimination constraints*. Despite its exponential number of constraints, the formulation (2.15) is very effective in practice. We will return to this formulation in Chap. 7.

Kaibel and Weltge [224] show that the traveling salesman problem cannot be formulated with polynomially many inequalities in the space of variables  $x_e$ ,  $e \in E$ .

## 2.8 The Generalized Assignment Problem

The *generalized assignment problem* is the following 0,1 program, defined by coefficients  $c_{ij}$  and  $t_{ij}$ , and capacities  $T_j$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, n$ ,

$$\begin{aligned}
 \max \quad & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\
 & \sum_{j=1}^n x_{ij} = 1 \quad i = 1, \dots, m \\
 & \sum_{i=1}^m t_{ij} x_{ij} \leq T_j \quad j = 1, \dots, n \\
 & x \in \{0, 1\}^{m \times n}.
 \end{aligned} \tag{2.16}$$

The following example is a variation of this model. In hospitals, operating rooms are a scarce resource that needs to be utilized optimally. The basic problem can be formulated as follows, acknowledging that each hospital will have its own specific additional constraints. Suppose that a hospital has  $n$  operating rooms. During a given time period  $T$ , there may be  $m$  surgeries that could potentially be scheduled. Let  $t_{ij}$  be the estimated time of operating on patient  $i$  in room  $j$ , for  $i = 1, \dots, m$ ,  $j = 1, \dots, n$ . The goal is to schedule surgeries during the given time period so as to waste as little of the operating rooms' capacity as possible.

Let  $x_{ij}$  be a binary variable that takes the value 1 if patient  $i$  is operated on in operating room  $j$ , and 0 otherwise. The basic operating rooms scheduling problem is as follows:

$$\begin{aligned}
 \max \quad & \sum_{i=1}^m \sum_{j=1}^n t_{ij} x_{ij} \\
 & \sum_{j=1}^n x_{ij} \leq 1 \quad i = 1, \dots, m \\
 & \sum_{i=1}^m t_{ij} x_{ij} \leq T \quad j = 1, \dots, n \\
 & x \in \{0, 1\}^{m \times n}.
 \end{aligned} \tag{2.17}$$

The objective is to maximize the utilization time of the operating rooms during the given time period (this is equivalent to minimizing wasted capacity). The first constraints guarantee that each patient  $i$  is operated on at most once. If patient  $i$  *must* be operated on during this period, the inequality constraint is changed into an equality. The second constraints are the capacity constraints on each of the operating rooms.

A special case of interest is when all operating rooms are identical, that is,  $t_{ij} := t_i$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, n$ , where the estimated time  $t_i$  of operation  $i$  is independent of the operating room. In this case, the above formulation admits numerous symmetric solutions, since permuting operating rooms does not modify the objective value. Intuitively, symmetry in the

problem seems helpful but, in fact, it may cause difficulties in the context of a standard branch-and-cut algorithm. This is due to the creation of a potentially very large number of isomorphic subproblems in the enumeration tree, resulting in a duplication of the computing effort unless the isomorphisms are discovered. Special techniques are available to deal with symmetries, such as isomorphism pruning, which can be incorporated in branch-and-cut algorithms. We will discuss this in Chap. 9.

The operating room scheduling problem is often complicated by the fact that there is also a limited number of surgeons, each surgeon can only perform certain operations, and a support team (anesthesiologist, nurses) needs to be present during the operation. To deal with these aspects of the operating room scheduling problem, one needs new variables and constraints.

## 2.9 The Mixing Set

We now describe a mixed integer linear set associated with a simple make-or-buy problem. The demand for a given product takes values  $b_1, \dots, b_n \in \mathbb{R}$  with probabilities  $p_1, \dots, p_n$ . Note that the demand values in this problem need not be integer. Today we produce an amount  $y \in \mathbb{R}$  of the product at a unit cost  $h$ , before knowing the actual demand. Tomorrow the actual demand  $b_i$  is experienced; if  $b_i > y$  then we purchase the extra amount needed to meet the demand at a unit cost  $c$ . However, the product can only be purchased in unit batches, that is, in integer amounts. The problem is to describe the production strategy that minimizes the expected total cost. Let  $x_i$  be the amount purchased tomorrow if the demand takes value  $b_i$ . Define the mixing set

$$MIX := \{(y, x) \in \mathbb{R}_+ \times \mathbb{Z}_+^n : y + x_i \geq b_i, 1 \leq i \leq n\}.$$

Then the above problem can be formulated as

$$\begin{aligned} \min \quad & hy + c \sum_{i=1}^n p_i x_i \\ & (y, x) \in MIX. \end{aligned}$$

## 2.10 Modeling Fixed Charges

Integer variables naturally represent entities that come in discrete amounts. They can also be used to model:

- logical conditions such as implications or dichotomies;
- nonlinearities, such as piecewise linear functions;
- nonconvex sets that can be expressed as a union of polyhedra.

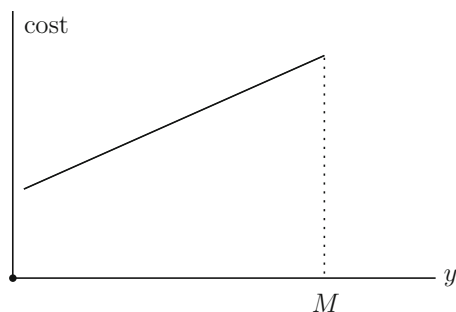


Figure 2.4: Fixed and variable costs

We introduce some of these applications. Economic activities frequently involve both fixed and variable costs. In this case, the cost associated with a certain variable  $y$  is 0 when the variable  $y$  takes value 0, and it is  $c + hy$  whenever  $y$  takes positive value (see Fig. 2.4). For example, variable  $y$  may represent a production quantity that incurs both a fixed cost if anything is produced at all (e.g., for setting up the machines), and a variable cost (e.g., for operating the machines). This situation can be modeled using a binary variable  $x$  indicating whether variable  $y$  takes a positive value. Let  $M$  be some upper bound, known a priori, on the value of variable  $y$ . The (nonlinear) cost of variable  $y$  can be written as the linear expression

$$cx + hy$$

where we impose

$$\begin{aligned} y &\leq Mx \\ x &\in \{0, 1\} \\ y &\geq 0. \end{aligned}$$

Such “big  $M$ ” formulations should be used with caution in integer programming because their linear programming relaxations tend to produce weak bounds in branch-and-bound algorithms. Whenever possible, one should use the tightest known bound, instead of an arbitrarily large  $M$ . We give two examples.

### 2.10.1 Facility Location

A company would like to set up facilities in order to serve geographically dispersed customers at minimum cost. The  $m$  customers have known annual demands  $d_i$ , for  $i = 1, \dots, m$ . The company can open a facility of capacity  $u_j$  and fixed annual operating cost  $f_j$  in location  $j$ , for  $j = 1, \dots, n$ . Knowing

the variable cost  $c_{ij}$  of transporting one unit of goods from location  $j$  to customer  $i$ , where should the company locate its facilities in order to minimize its annual cost

To formulate this problem, we introduce variables  $x_j$  that take the value 1 if a facility is opened in location  $j$ , and 0 if not. Let  $y_{ij}$  be the fraction of the demand  $d_i$  transported annually from  $j$  to  $i$ .

$$\begin{aligned}
 \min \quad & \sum_{i=1}^m \sum_{j=1}^n c_{ij} d_i y_{ij} + \sum_{j=1}^n f_j x_j \\
 & \sum_{j=1}^n y_{ij} = 1 \quad i = 1, \dots, m \\
 & \sum_{i=1}^m d_i y_{ij} \leq u_j x_j \quad j = 1, \dots, n \\
 & y \geq 0 \\
 & x \in \{0, 1\}^n.
 \end{aligned}$$

The objective function is the total yearly cost (transportation plus operating costs). The first set of constraints guarantees that the demand is met, the second type of constraints are capacity constraints at the facilities. Note that the capacity constraints are fixed charge constraints, since they force  $x_j = 1$  whenever  $y_{ij} > 0$  for some  $i$ .

A classical special case is the uncapacitated facility location problem, in which  $u_j = +\infty$ ,  $j = 1, \dots, n$ . In this case, it is always optimal to satisfy all the demand of client  $i$  from the closest open facility, therefore  $y_{ij}$  can be assumed to be binary. Hence the problem can be formulated as

$$\begin{aligned}
 \min \quad & \sum \sum c_{ij} d_i y_{ij} + \sum f_j x_j \\
 & \sum_j y_{ij} = 1 \quad i = 1, \dots, m \\
 & \sum_i y_{ij} \leq m x_j \quad j = 1, \dots, n \\
 & y \in \{0, 1\}^{m \times n}, \quad x \in \{0, 1\}^n.
 \end{aligned} \tag{2.18}$$

Note that the constraint  $\sum_i y_{ij} \leq m x_j$  forces  $x_j = 1$  whenever  $y_{ij} > 0$  for some  $i$ . The same condition could be enforced by the disaggregated set of constraints  $y_{ij} \leq x_j$ , for all  $i, j$ .

$$\begin{aligned}
 \min \quad & \sum \sum c_{ij} d_i y_{ij} + \sum f_j x_j \\
 & \sum_j y_{ij} = 1 \quad i = 1, \dots, m \\
 & y_{ij} \leq x_j \quad i = 1, \dots, m, j = 1, \dots, n \\
 & y \in \{0, 1\}^{m \times n}, \quad x \in \{0, 1\}^n.
 \end{aligned} \tag{2.19}$$



The disaggregated formulation (2.19) is stronger than the aggregated one (2.18), since the constraint  $\sum_i y_{ij} \leq mx_i$  is just the sum of the constraints  $y_{ij} \leq x_i$ ,  $i = 1, \dots, m$ . According to the paradigm presented in Sect. 2.2 in this chapter, the disaggregated formulation is better, because it yields tighter bounds in a branch-and-cut algorithm. In practice it has been observed that the difference between these two bounds is typically enormous. It is natural to conclude that formulation (2.19) is the one that should be used in practice. However, the situation is more complicated. When the aggregated formulation (2.18) is given to state-of-the-art solvers, they are able to detect and generate disaggregated constraints on the fly, whenever these constraints are violated by the current feasible solution. So, in fact, it is preferable to use the aggregated formulation because the size of the linear relaxation is much smaller and faster to solve.

Let us elaborate on this interesting point. Nowadays, state-of-the-art solvers automatically detect violated minimal cover inequalities (this notion was introduced in Sect. 2.2), and the disaggregated constraints in (2.19) happen to be minimal cover inequalities for the aggregated constraints. More formally, let us write the aggregated constraint relative to facility  $j$  as

$$mz_j + \sum_{i=1}^m y_{ij} \leq m$$

where  $z_j = 1 - x_j$  is also a 0, 1 variable. This is a knapsack constraint. Note that any minimal cover inequality is of the form  $z_j + y_{ij} \leq 1$ . Substituting  $1 - x_j$  for  $z_j$ , we get the disaggregated constraint  $y_{ij} \leq x_j$ . We will discuss the separation of minimal cover inequalities in Sect. 7.1.

### 2.10.2 Network Design

Network design problems arise in the telecommunication industry. Let  $N$  be a given set of nodes. Consider a directed network  $G = (N, A)$  consisting of arcs that could be constructed. We need to select a subset of arcs from  $A$  in order to route commodities. Commodity  $k$  has a source  $s_k \in N$ , a destination  $t_k \in N$ , and volume  $v_k$  for  $k = 1, \dots, K$ . Each commodity can be viewed as a flow that must be routed through the network. Each arc  $a \in A$  has a construction cost  $f_a$  and a capacity  $c_a$ . If we select arc  $a$ , the sum of the commodity flows going through arc  $a$  should not exceed its capacity  $c_a$ . Of course, if we do not select arc  $a$ , no flow can be routed through  $a$ . How should we design the network in order to route all the demand at minimum cost?

Let us introduce binary variables  $x_a$ , for  $a \in A$ , where  $x_a = 1$  if arc  $a$  is constructed, 0 otherwise. Let  $y_a^k$  denote the amount of commodity  $k$  flowing through arc  $a$ . The formulation is

$$\begin{aligned}
 & \min \quad \sum_{a \in A} f_a x_a \\
 & \sum_{a \in \delta^+(i)} y_{ij}^k - \sum_{a \in \delta^-(i)} y_{ji}^k = \begin{cases} v_k & \text{for } i = s_k \\ -v_k & \text{for } i = t_k \\ 0 & \text{for } i \in N \setminus \{s_k, t_k\} \end{cases} \quad \text{for } k = 1, \dots, K \\
 & \sum_{k=1}^K y_a^k \leq c_a x_a \quad \text{for } a \in A \\
 & y \geq 0 \\
 & x_a \in \{0, 1\} \quad \text{for } a \in A.
 \end{aligned}$$

The first set of constraints are conservation of flow constraints: For each commodity  $k$ , the amount of flow out of node  $i$  equals to the amount of flow going in, except at the source and destination. The second constraints are the capacity constraints that need to be satisfied for each arc  $a \in A$ . Note that they are fixed-charge constraints.

## 2.11 Modeling Disjunctions

Many applications have disjunctive constraints. For example, when scheduling jobs on a machine, we might need to model that either job  $i$  is scheduled before job  $j$  or vice versa; if  $p_i$  and  $p_j$  denote the processing times of these two jobs on the machine, we then need a constraint stating that the starting times  $t_i$  and  $t_j$  of jobs  $i$  and  $j$  satisfy  $t_j \geq t_i + p_i$  or  $t_i \geq t_j + p_j$ . In such applications, the feasible solutions lie in the union of two or more polyhedra.

In this section, the goal is to model that a point belongs to the union of  $k$  polytopes in  $\mathbb{R}^n$ , namely bounded sets of the form

$$\begin{aligned}
 & A_i y \leq b_i \\
 & 0 \leq y \leq u_i,
 \end{aligned} \tag{2.20}$$

for  $i = 1, \dots, k$ . The same modeling question is more complicated for unbounded polyhedra and will be discussed in Sect. 4.9.

A way to model the union of  $k$  polytopes in  $\mathbb{R}^n$  is to introduce  $k$  variables  $x_i \in \{0, 1\}$ , indicating whether  $y$  is in the  $i$ th polytope, and  $k$  vectors of variables  $y_i \in \mathbb{R}^n$ . The vector  $y \in \mathbb{R}^n$  belongs to the union of the  $k$  polytopes (2.20) if and only if

$$\begin{aligned}
\sum_{i=1}^k y_i &= y \\
A_i y_i &\leq b_i x_i & i = 1, \dots, k \\
0 \leq y_i &\leq u_i x_i & i = 1, \dots, k \\
\sum_{i=1}^k x_i &= 1 \\
x &\in \{0, 1\}^k.
\end{aligned} \tag{2.21}$$

The next proposition shows that formulation (2.21) is perfect in the sense that the convex hull of its solutions is simply obtained by dropping the integrality restriction.

**Proposition 2.6.** *The convex hull of solutions to (2.21) is*

$$\begin{aligned}
\sum_{i=1}^k y_i &= y \\
A_i y_i &\leq b_i x_i & i = 1, \dots, k \\
0 \leq y_i &\leq u_i x_i & i = 1, \dots, k \\
\sum_{i=1}^k x_i &= 1 \\
x &\in [0, 1]^k.
\end{aligned}$$

*Proof.* Let  $P \subset \mathbb{R}^n \times \mathbb{R}^{kn} \times \mathbb{R}^k$  be the polytope given in the statement of the proposition. It suffices to show that any point  $\bar{z} := (\bar{y}, \bar{y}_1, \dots, \bar{y}_k, \bar{x}_1, \dots, \bar{x}_k)$  in  $P$  is a convex combination of solutions to (2.21). For  $t$  such that  $\bar{x}_t \neq 0$ , define the point  $z^t = (y^t, y_1^t, \dots, y_k^t, x_1^t, \dots, x_k^t)$  where

$$y^t := \frac{\bar{y}_t}{\bar{x}_t}, \quad y_i^t := \begin{cases} \frac{\bar{y}_t}{\bar{x}_t} & \text{for } i = t, \\ 0 & \text{otherwise,} \end{cases} \quad x_i^t := \begin{cases} 1 & \text{for } i = t, \\ 0 & \text{otherwise.} \end{cases}$$

The  $z^t$ s are solutions of (2.21). We claim that  $\bar{z}$  is a convex combination of these points, namely  $\bar{z} = \sum_{t: \bar{x}_t \neq 0} \bar{x}_t z^t$ . To see this, observe first that  $\bar{y} = \sum \bar{y}_i = \sum_{t: \bar{x}_t \neq 0} \bar{y}_t = \sum_{t: \bar{x}_t \neq 0} \bar{x}_t y^t$ . Second, note that when  $\bar{x}_i \neq 0$  we have  $\bar{y}_i = \sum_{t: \bar{x}_t \neq 0} \bar{x}_t y_i^t$ . This equality also holds when  $\bar{x}_i = 0$  because then  $\bar{y}_i = 0$  and  $y_i^t = 0$  for all  $t$  such that  $\bar{x}_t \neq 0$ . Finally  $\bar{x}_i = \sum_{t: \bar{x}_t \neq 0} \bar{x}_t x_i^t$  for  $i = 1, \dots, k$ .  $\square$

## 2.12 The Quadratic Assignment Problem and Fortet's Linearization

In this book we mostly deal with *linear* integer programs. However, *non-linear* integer programs (in which the objective function or some of the constraints defining the feasible region are nonlinear) are important in some applications. The *quadratic assignment problem (QAP)* is an example of a nonlinear 0,1 program that is simple to state but notoriously difficult to solve. Interestingly, we will show that it can be linearized.

We have to place  $n$  facilities in  $n$  locations. The data are the amount  $f_{k\ell}$  of goods that has to be shipped from facility  $k$  to facility  $\ell$ , for  $k = 1, \dots, n$  and  $\ell = 1, \dots, n$ , and the distance  $d_{ij}$  between locations  $i, j$ , for  $i = 1, \dots, n$  and  $j = 1, \dots, n$ .

The problem is to assign facilities to locations so as to minimize the total cumulative distance traveled by the goods. For example, in the electronics industry, the quadratic assignment problem is used to model the problem of placing interconnected electronic components onto a microchip or a printed circuit board.

Let  $x_{ki}$  be a binary variable that takes the value 1 if facility  $k$  is assigned to location  $i$ , and 0 otherwise. The quadratic assignment problem can be formulated as follows:

$$\begin{aligned} \max \quad & \sum_{i,j} \sum_{k,\ell} d_{ij} f_{k\ell} x_{ki} x_{\ell j} \\ & \sum_k x_{ki} = 1 & i = 1, \dots, n \\ & \sum_i x_{ki} = 1 & k = 1, \dots, n \\ & x \in \{0, 1\}^{n \times n}. \end{aligned}$$

The quadratic assignment problem is an example of a *0,1 polynomial program*

$$\begin{aligned} \min \quad & z = f(x) \\ & g_i(x) = 0 & i = 1, \dots, m \\ & x_j \in \{0, 1\} & j = 1, \dots, n \end{aligned} \tag{2.22}$$

where the functions  $f$  and  $g_i$  ( $i = 1, \dots, m$ ) are polynomials. Fortet [144] observed that such nonlinear functions can be linearized when the variables only take value 0 or 1.

**Proposition 2.7.** *Any 0,1 polynomial program (2.22) can be formulated as a pure 0,1 linear program by introducing additional variables.*

*Proof.* Note that, for any integer exponent  $k \geq 1$ , the 0,1 variable  $x_j$  satisfies  $x_j^k = x_j$ . Therefore we can replace each expression of the form  $x_j^k$  with  $x_j$ , so that no variable appears in  $f$  or  $g_i$  with exponent greater than 1.

The product  $x_i x_j$  of two 0,1 variables can be replaced by a new 0,1 variable  $y_{ij}$  related to  $x_i, x_j$  by linear constraints. Indeed, to guarantee that  $y_{ij} = x_i x_j$  when  $x_i$  and  $x_j$  are binary variables, it suffices to impose the linear constraints  $y_{ij} \leq x_i$ ,  $y_{ij} \leq x_j$  and  $y_{ij} \geq x_i + x_j - 1$  in addition to the 0,1 conditions on  $x_i, x_j, y_{ij}$ .  $\square$

As an example, consider  $f$  defined by  $f(x) = x_1^5 x_2 + 4x_1 x_2 x_3^2$ . Applying Fortet's linearization sequentially, function  $f$  is initially replaced by  $z = x_1 x_2 + 4x_1 x_2 x_3$  for 0,1 variables  $x_j$ ,  $j = 1, 2, 3$ . Subsequently, we introduce 0,1 variables  $y_{12}$  in place of  $x_1 x_2$ , and  $y_{123}$  in place of  $y_{12} x_3$ , so that the objective function is replaced by the linear function  $z = y_{12} + 4y_{123}$ , where we impose

$$\begin{aligned} y_{12} &\leq x_1, & y_{12} &\leq x_2, & y_{12} &\geq x_1 + x_2 - 1, \\ y_{123} &\leq y_{12}, & y_{123} &\leq x_3, & y_{123} &\geq y_{12} + x_3 - 1, \\ y_{12}, y_{123}, x_1, x_2, x_3 &\in \{0, 1\}. \end{aligned}$$

## 2.13 Further Readings

The book “Applications of Optimization with Xpress” by Gu  ret, Prins, and Servaux [193], which can also be downloaded online, provides an excellent guide for constructing integer programming formulations in various areas such as planning, transportation, telecommunications, economics, and finance. The book “Production Planning by Mixed-Integer Programming” by Pochet and Wolsey [309] contains several optimization models in production planning and an accessible exposition of the theory of mixed integer linear programming. The book “Optimization Methods in Finance” by Cornu  jols and T  t  nc   [96] gives an application of integer programming to modeling index funds. Several formulations in this chapter are defined on graphs. We refer to Bondy and Murty [62] for a textbook on graph theory.

The knapsack problem is one of the most widely studied models in integer programming. A classic book for the knapsack problem is the one of Martello and Toth [268], which is downloadable online. A more recent textbook is [234]. In Sect. 2.2 we introduced alternative formulations (in the context of 0,1 knapsack set) and discussed the strength of different formulations. This topic is central in integer programming theory and applications. In fact, a strong formulation is a key ingredient to solving integer programs

even of moderate size: A weak formulation may prove to be unsolvable by state-of-the-art solvers even for small-size instances. Formulations can be strengthened a priori or dynamically, by adding cuts and this will be discussed at length in this book. Strong formulations can also be obtained with the use of additional variables, that model properties of a mixed integer set to be optimized and we will develop this topic. The book “Integer Programming” by Wolsey [353] contains an accessible exposition of this topic.

There is a vast literature on the traveling salesman problem: This problem is easy to state and it has been popular for testing the methods exposed in this book. The book edited by Lawler, Lenstra, Rinnooy Kan, and Shmoys [253] contains a series of important surveys; for instance the chapters on polyhedral theory and computations by Grötschel and Padberg. The book by Applegate, Bixby, Chvátal, and Cook [13] gives a detailed account of the theory and the computational advances that led to the solution of traveling salesman instances of enormous size. The recent book “In the pursuit of the traveling salesman” by Cook [86] provides an entertaining account of the traveling salesman problem, with many historical insights.

Vehicle routing is related to the traveling salesman problem and refers to a class of problems where goods located at a central depot need to be delivered to customers who have placed orders for such goods. The goal is to minimize the cost of delivering the goods. There are many references in this area. We just cite the monograph of Toth and Vigo [337].

Constraint programming has been mentioned while introducing formulations for the Sudoku game. The interaction between integer programming and constraint programming is a growing area of research, see, e.g., Hooker [206] and Achterberg [5].

For machine scheduling we mention the survey of Queyranne and Schulz [312].

## 2.14 Exercises

**Exercise 2.1.** Let

$$S := \{x \in \{0, 1\}^4 : 90x_1 + 35x_2 + 26x_3 + 25x_4 \leq 138\}.$$

(i) Show that

$$S = \{x \in \{0, 1\}^4 : 2x_1 + x_2 + x_3 + x_4 \leq 3\},$$

and

$$S = \{x \in \{0, 1\}^4 : \begin{array}{rrrrr} 2x_1 & +x_2 & +x_3 & +x_4 & \leq 3 \\ & x_1 & +x_2 & +x_3 & \leq 2 \\ & x_1 & +x_2 & & +x_4 \leq 2 \\ & & x_1 & +x_3 & +x_4 \leq 2 \end{array}\}.$$

- (ii) Can you rank these three formulations in terms of the tightness of their linear relaxations, when  $x \in \{0, 1\}^4$  is replaced by  $x \in [0, 1]^4$ ? Show any strict inclusion.

**Exercise 2.2.** Give an example of a 0, 1 knapsack set where both  $P \setminus P^C \neq \emptyset$  and  $P^C \setminus P \neq \emptyset$ , where  $P$  and  $P^C$  are the linear relaxations of the knapsack and minimal cover formulations respectively.

**Exercise 2.3.** Produce a family of 0, 1 knapsack sets (having an increasing number  $n$  of variables) whose associated family of minimal covers grows exponentially with  $n$ .

**Exercise 2.4.** (Constraint aggregation) Given a finite set  $E$  and a clutter  $\mathcal{C}$  of subsets of  $E$ , does there always exist a 0, 1 knapsack set  $K$  such that  $\mathcal{C}$  is the family of all minimal covers of  $K$ ? Prove or disprove.

**Exercise 2.5.** Show that any integer linear program of the form

$$\begin{array}{ll} \min & cx \\ & Ax = b \\ & 0 \leq x \leq u \\ & x \text{ integral} \end{array}$$

can be converted into a 0, 1 knapsack problem.

**Exercise 2.6.** The pigeonhole principle states that the problem

- (P) Place  $n + 1$  pigeons into  $n$  holes so that no two pigeons share a hole has no solution.

Formulate (P) as an integer linear program with two kinds of constraints:

- (a) those expressing the condition that every pigeon must get into a hole;
- (b) those expressing the condition that, for each pair of pigeons, at most one of the two birds can get into a given hole.

Show that there is no integer solution satisfying (a) and (b), but that the linear program with constraints (a) and (b) is feasible.

**Exercise 2.7.** Let  $A$  be a  $0,1$  matrix and let  $A^{max}$  be the row submatrix of  $A$  containing one copy of all the rows of  $A$  whose support is not included in the support of another row of  $A$ . Show that the packing sets  $S^P(A)$  and  $S^P(A^{max})$  coincide and that their linear relaxations are equivalent.

Similarly let  $A^{min}$  be the row submatrix of  $A$  containing one copy of all the rows of  $A$  whose support does not include the support of another row of  $A$ . Show that  $S^C(A)$  and  $S^C(A^{min})$  coincide and that their linear relaxations are equivalent.

**Exercise 2.8.** We use the notation introduced in Sect. 2.4.2. Given the matrix

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

- What is  $G_A$ ?
- What is  $A_c$ ?
- Give a formulation for  $S^P(A)$  that is better than  $A_c x \leq 1, 0 \leq x \leq 1$ .

**Exercise 2.9.** Let  $A$  be a matrix with two 1's per row. Show that the sets  $S^P(A)$  and  $S^C(A)$  have the same cardinality.

**Exercise 2.10.** Given a clutter  $\mathcal{F}$ , let  $A$  be the incidence matrix of the family  $\mathcal{F}$  and  $G_A$  the intersection graph of  $A$ . Prove that  $A$  is the clique matrix of  $G_A$  if and only if the following holds:

For every  $F_1, F_2, F_3$  in  $\mathcal{F}$ , there is an  $F \in \mathcal{F}$  that contains  $(F_1 \cap F_2) \cup (F_1 \cap F_3) \cup (F_2 \cap F_3)$ .

**Exercise 2.11.** Let  $T$  be a minimal transversal of  $\mathcal{S}$  and  $e_j \in T$ . Then  $T \cap S_i = \{e_j\}$  for some  $S_i \in \mathcal{S}$ .

**Exercise 2.12.** Prove that, for an undirected connected graph  $G = (V, E)$ , the following pairs of families of subsets of edges of  $G$  are blocking pairs:



- Spanning trees and minimal cuts
- $st$ -paths and minimal  $st$ -cuts.
- Minimal postman sets and minimal odd cuts. (A set  $E' \subseteq E$  is a *postman set* if  $G = (V, E \setminus E')$  is an Eulerian graph and a cut is *odd* if it contains an odd number of edges.) Assume that  $G$  is not an Eulerian graph.

**Exercise 2.13.** Construct an example showing that the formulation (2.4) is not perfect.

**Exercise 2.14.** Show that a graph is bipartite if and only if it contains no odd cycle.

**Exercise 2.15.** (Chromatic number) The following is (a simplified version of) a *frequency assignment problem* in telecommunications. Transmitters  $1, \dots, n$  broadcast different signals using preassigned frequencies. Transmitters that are geographically close might interfere and they must therefore use distinct frequencies. The problem is to determine the minimum number of frequencies that need to be assigned to the transmitters so that interference is avoided.

This problem has a natural graph-theoretic counterpart: The *chromatic number*  $\chi(G)$  of an undirected graph  $G = (V, E)$  is the minimum number of colors to be assigned to the nodes of  $G$  so that adjacent nodes receive distinct colors. Equivalently, the chromatic number is the minimum number of (maximal) stable sets whose union is  $V$ .

Define the *interference graph* of a frequency assignment problem to be the undirected graph  $G = (V, E)$  where  $V$  represents the set of transmitters and  $E$  represents the set of pairs of transmitters that would interfere with each other if they were assigned the same frequency. Then the minimum number of frequencies to be assigned so that interference is avoided is the chromatic number of the interference graph.

Consider the following integer programs. Let  $\mathcal{S}$  be the family of all maximal stable sets of  $G$ . The first one has one variable  $x_S$  for each maximal stable set  $S$  of  $G$ , where  $x_S = 1$  if  $S$  is used as a color,  $x_S = 0$  otherwise.

$$\begin{aligned} \chi_1(G) = \min \sum_{S \in \mathcal{S}} x_S \\ \sum_{S \supseteq \{v\}} x_S \geq 1 \quad v \in V \\ x_S \in \{0, 1\} \quad S \in \mathcal{S}. \end{aligned}$$

The second one has one variable  $x_{v,c}$  for each node  $v$  in  $V$  and color  $c$  in a set  $C$  of available colors (with  $|C| \geq \chi(G)$ ), where  $x_{v,c} = 1$  if color  $c$  is assigned to node  $v$ , 0 otherwise. It also has color variables,  $y_c = 1$  if color  $c$  is used, 0 otherwise.

$$\begin{aligned}\chi_2(G) &= \min \sum_{c \in C} y_c \\ x_{u,c} + x_{v,c} &\leq 1 \quad \forall uv \in E \text{ and } c \in C \\ x_{v,c} &\leq y_c \quad v \in V, c \in C \\ \sum_{c \in C} x_{v,c} &= 1 \quad v \in V \\ x_{v,c} &\in \{0, 1\}, y_c \geq 0 \quad v \in V, c \in C.\end{aligned}$$

- Show that  $\chi_1(G) = \chi_2(G) = \chi(G)$ .
- Let  $\chi_1^*(G)$ ,  $\chi_2^*(G)$  be the optimal values of the linear programming relaxations of the above integer programs. Prove that  $\chi_1^*(G) \geq \chi_2^*(G)$  for all graphs  $G$ . Prove that  $\chi_1^*(G) > \chi_2^*(G)$  for some graph  $G$ .

**Exercise 2.16** (Combinatorial auctions). A company sets an auction for  $N$  objects. Bidders place their bids for some subsets of the  $N$  objects that they like. The auction house has received  $n$  bids, namely bids  $b_j$  for subset  $S_j$ , for  $j = 1, \dots, n$ . The auction house is faced with the problem of choosing the winning bids so that profit is maximized and each of the  $N$  objects is given to at most one bidder. Formulate the optimization problem faced by the auction house as a set packing problem.

**Exercise 2.17.** (Single machine scheduling) Jobs  $\{1, \dots, n\}$  must be processed on a single machine. Each job is available for processing after a certain time, called release time. For each job we are given its release time  $r_i$ , its processing time  $p_i$  and its weight  $w_i$ . Formulate as an integer linear program the problem of sequencing the jobs without overlap or interruption so that the sum of the weighted completion times is minimized.

**Exercise 2.18.** (Lot sizing) The demand for a product is known to be  $d_t$  units in periods  $t = 1, \dots, n$ . If we produce the product in period  $t$ , we incur a machine setup cost  $f_t$  which does not depend on the number of units produced plus a production cost  $p_t$  per unit produced. We may produce any number of units in any period. Any inventory carried over from period  $t$  to period  $t + 1$  incurs an inventory cost  $i_t$  per unit carried over. Initial inventory is  $s_0$ . Formulate a mixed integer linear program in order to meet the demand over the  $n$  periods while minimizing overall costs.

**Exercise 2.19.** A firm is considering project  $A, B, \dots, H$ . Using binary variables  $x_a, \dots, x_h$  and linear constraints, model the following conditions on the projects to be undertaken.

1. At most one of  $A, B, \dots, H$ .
2. Exactly two of  $A, B, \dots, H$ .
3. If  $A$  then  $B$ .
4. If  $A$  then not  $B$ .
5. If not  $A$  then  $B$ .
6. If  $A$  then  $B$ , and if  $B$  then  $A$ .
7. If  $A$  then  $B$  and  $C$ .
8. If  $A$  then  $B$  or  $C$ .
9. If  $B$  or  $C$  then  $A$ .
10. If  $B$  and  $C$  then  $A$ .
11. If two or more of  $B, C, D, E$  then  $A$ .
12. If  $m$  or more than  $n$  projects  $B, \dots, H$  then  $A$ .

**Exercise 2.20.** Prove or disprove that the formulation  $\mathcal{F} = \{x \in \{0, 1\}^{n^2}, \sum_{i=1}^n x_{ij} = 1 \text{ for } 1 \leq j \leq n, \sum_{j=1}^n x_{ij} = 1 \text{ for } 1 \leq i \leq n\}$  describes the set of  $n \times n$  permutation matrices.

**Exercise 2.21.** For the following subsets of edges of an undirected graph  $G = (V, E)$ , find an integer linear formulation and prove its correctness:

- The family of Hamiltonian paths of  $G$  with endnodes  $u, v$ . (A *Hamiltonian path* is a path that goes exactly once through each node of the graph.)
- The family of all Hamiltonian paths of  $G$ .
- The family of edge sets that induce a triangle of  $G$ .
- Assuming that  $G$  has  $3n$  nodes, the family of  $n$  node-disjoint triangles.
- The family of odd cycles of  $G$ .

**Exercise 2.22.** Consider a connected undirected graph  $G = (V, E)$ . For  $S \subseteq V$ , denote by  $E(S)$  the set of edges with both ends in  $S$ . For  $i \in V$ , denote by  $\delta(i)$  the set of edges incident with  $i$ . Prove or disprove that the following formulation produces a spanning tree with maximum number of leaves.

$$\begin{aligned} \max \quad & \sum_{i \in V} z_i \\ \sum_{e \in E} x_e = \quad & |V| - 1 \\ \sum_{e \in E(S)} x_e \leq \quad & |S| - 1 \quad S \subset V, |S| \geq 2 \\ \sum_{e \in \delta(i)} x_e + (|\delta(i)| - 1)z_i \leq \quad & |\delta(i)| \quad i \in V \\ x_e \in \quad & \{0, 1\} \quad e \in E \\ z_i \in \quad & \{0, 1\} \quad i \in V. \end{aligned}$$

**Exercise 2.23.** One sometimes would like to maximize the sum of nonlinear functions  $\sum_{i=1}^n f_i(x_i)$  subject to  $x \in P$ , where  $f_i : \mathbb{R} \rightarrow \mathbb{R}$  for  $i = 1, \dots, n$  and  $P$  is a polytope. Assume  $P \subset [l, u]$  for  $l, u \in \mathbb{R}^n$ . Show that, if the functions  $f_i$  are piecewise linear, this problem can be formulated as a mixed integer linear program. For example a utility function might be approximated by  $f_i$  as shown in Fig. 2.5 (risk-averse individuals dislike more a monetary loss of  $y$  than they like a monetary gain of  $y$  dollars).

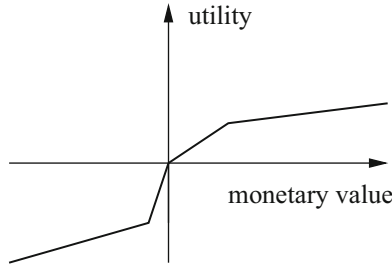


Figure 2.5: Example of a piecewise linear utility function

**Exercise 2.24.**

- (i) Write the logic statement  $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg(x_1 \wedge x_2) \wedge x_3)$  in conjunctive normal form.
- (ii) Formulate the following logical inference problem as an integer linear program. “Does the proposition  $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg(x_1 \wedge x_2) \wedge x_3)$  imply  $x_1 \vee x_2 \vee x_3$ ?”

**Exercise 2.25.** Let  $x_1, \dots, x_n$  be atomic propositions and let  $A$  and  $B$  be two logic statements in CNF. The logic statement  $A \implies B$  is satisfied if any truth assignment that satisfies  $A$  also satisfies  $B$ . Prove that  $A \implies B$  is satisfied if and only if the logic statement  $\neg A \vee B$  is satisfied.

**Exercise 2.26.** Consider a 0,1 set  $S := \{x \in \{0,1\}^n : Ax \leq b\}$  where  $A \in \mathbb{R}^{m \times n}$  and  $b \in \mathbb{R}^m$ . Prove that  $S$  can be written in the form  $S = \{x \in \{0,1\}^n : Dx \leq d\}$  where  $D$  is a matrix all of whose entries are 0, +1 or -1 (Matrices  $D$  and  $A$  may have a different number of rows).

**Exercise 2.27** (Excluding  $(0,1)$ -vectors). Find integer linear formulations for the following integer sets (Hint: Use the generalized set covering inequalities).

- The set of all  $(0,1)$ -vectors in  $\mathbb{R}^4$  except  $\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$ .

- The set of all  $(0,1)$ -vectors in  $\mathbb{R}^6$  except  $\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$ .

- The set of all  $(0,1)$ -vectors in  $\mathbb{R}^6$  except all the vectors having exactly two 1s in the first 3 components and one 1 in the last 3 components.
- The set of all  $(0,1)$ -vectors in  $\mathbb{R}^n$  with an even number of 1s.
- The set of all  $(0,1)$ -vectors in  $\mathbb{R}^n$  with an odd number of 1's.

**Exercise 2.28.** Show that if  $P = \{x \in \mathbb{R}^n : Ax \leq b\}$  is such that  $P \cap \mathbb{Z}^n$  is the set of 0–1 vectors with an even number of 1's, then  $Ax \leq b$  contains at least  $2^{n-1}$  inequalities.

**Exercise 2.29.** Given a Sudoku game and a solution  $\bar{x}$ , formulate as an integer linear program the problem of certifying that  $\bar{x}$  is the unique solution.

**Exercise 2.30** (Crucipixel Game). Given a  $m \times n$  grid, the purpose of the game is to darken some of the cells so that in every row (resp. column) the darkened cells form distinct strings of the lengths and in the order prescribed by the numbers on the left of the row (resp. on top of the column).



**Exercise 2.33.** Assume  $c \in \mathbb{Z}^n$ ,  $A \in \mathbb{Z}^{m \times n}$ ,  $b \in \mathbb{Z}^m$ . Give a polynomial transformation of the 0,1 linear program

$$\begin{aligned} \max \quad & cx \\ & Ax \leq b \\ & x \in \{0, 1\}^n \end{aligned}$$

into a quadratic program

$$\begin{aligned} \max \quad & cx - Mx^T(1 - x) \\ & Ax \leq b \\ & 0 \leq x \leq 1, \end{aligned}$$

i.e., show how to choose the scalar  $M$  as a function of  $A$ ,  $b$  and  $c$  so that an optimal solution of the quadratic program is always an optimal solution of the 0,1 linear program (if any).



The authors working on Chap. 2



Giacomo Zambelli at the US border. Immigration Officer: What is the purpose of your trip? Giacomo: Visiting a colleague; I am a mathematician. Immigration Officer: What do mathematicians do? Giacomo: Sit in a chair and think.



Integer Programming

Conforti, M.; Cornuejols, G.; Zambelli, G.

2014, XII, 456 p. 75 illus., Hardcover

ISBN: 978-3-319-11007-3