In the table, we have ordered the projects with the highest ratio on the left. Therefore, using ratios to represent priorities, the intuitive rule would suggest adopting projects *P*4, then *P*5, and then continuing in order until the budget is consumed. If we followed that logic, we would adopt *P*4 and *P*5, but their expenditures of $36 million would leave no room in the budget for other projects, and the total NPV would be $4.4 million. However, by judiciously departing from the priority ordering, we can reject *P*5 and instead adopt *P*1 and *P*3, thus achieving a full use of the capital budget and a NPV of $6.8 million. The lumpiness of capital expenses means that we can't adopt fractional projects and follow our intuitive sense of priorities. Instead, we have to account for the implications of each combination of expenditures in terms of the budget remaining when we examine which combinations of projects make sense. The number of combinations is large enough that it becomes tedious to write down all the possibilities. In large problems, that task would be prohibitively time consuming, yet an integer programming model can provide us with optimal solutions readily.

The capital budgeting model illustrates decisions that are of the yes/no variety. In fact, all variables in the capital budgeting model are of this type. In other integer programming models, some of the variables might correspond to yes/no decisions (and thus to binary variables), while other variables resemble the familiar types of decisions that we have seen in other linear programming models.

## 6.3. SET COVERING

The covering model is one of the basic linear programming model types. As introduced in Chapter 2, the model contains covering decisions corresponding to variables that are assumed to be divisible. However, when the decisions are of the yes/no variety, we have a binary-choice version of the covering model known as the *set-covering problem*. To describe how this model might arise, consider the situation at the Metropolis Police Department.

### EXAMPLE 6.3 *Metropolis Police Department*

The police department in the city of Metropolis, has divided the city into 15 patrol sectors so patrol cars can respond quickly to service calls. Until recently, the streets have been patrolled overnight by 15 patrol cars, one in each sector. However, severe budget cuts have forced the city to eliminate some patrols. The chief of police has mandated that each sector should be covered by at least one unit located within the sector or in an adjacent sector.

The simplified map (Figure 6.9) depicts the 15 patrol sectors in the city. Any pair of sectors that share a boundary are considered to be adjacent. (Sectors 4 and 5 are adjacent, but not Sectors 3 and 5.) In addition, Sectors 7 and 14 are not accessible from each other because their boundary is the site of the Goose Pond Dam, while Sectors 9 and 13 are not mutually accessible due to the terrain of Moose Mountain, which is located at their boundary.

Having analyzed the map, the chief wants to know what number of patrol units will be required to provide service to the city's 15 sectors. ■

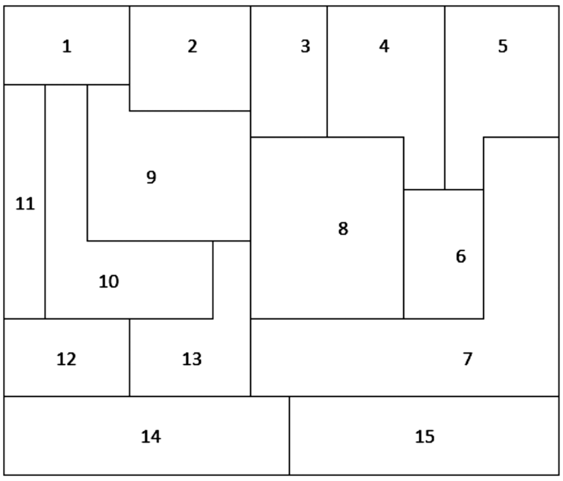


**Figure 6.9.** Sector map for Example 6.3.

Although the map is the basic source of data for this problem, we can use it to create an *adjacency array* for modeling purposes. This array is shown in Figure 6.10. In the figure, the numbered rows and columns correspond to the various sectors. An entry of 1 appears in row $i$ and column $j$ if sectors $i$ and $j$ are adjacent. (Each sector is considered adjacent to itself.)

A model for minimizing the required number of patrols can be constructed around the adjacency array by defining binary variables as follows.

$$\begin{aligned} y_j &= 1 \quad \text{if a patrol is assigned to sector } j \\ y_j &= 0 \quad \text{otherwise} \end{aligned}$$

Then the objective function—the total number of patrols—can be expressed as the sum of the $y_j$ values. In the model of Figure 6.11, this sum is represented as the SUMPRODUCT of the decision row and a row containing all 1s. To formulate constraints, suppose we focus on row $i$ in the array, which corresponds to sector $i$. The SUMPRODUCT of this row and the decisions yields the number of patrols that can service county $i$. In the problem, we require coverage from at least one patrol for every row. In words, our model takes the following form.

$$\begin{aligned} &\text{Minimize } z = \text{ the number of patrols} \\ &\text{subject to} \\ &\qquad\qquad \text{Patrols servicing sector } i \geq 1, \quad (\text{for all } i = 1, 2, \ldots, 15) \end{aligned}$$

In algebraic terms, we can let $a_{ij}$ represent the adjacency data in the array of Figure 6.10. That is, $a_{ij} = 1$ if sectors $i$ and $j$ are adjacent and $a_{ij} = 0$ if not. Then

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 8 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 9 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 10 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 11 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

**Figure 6.10.** Adjacency array for Example 6.3.

we can write

$$\text{Minimize } z = \sum y_j$$

subject to

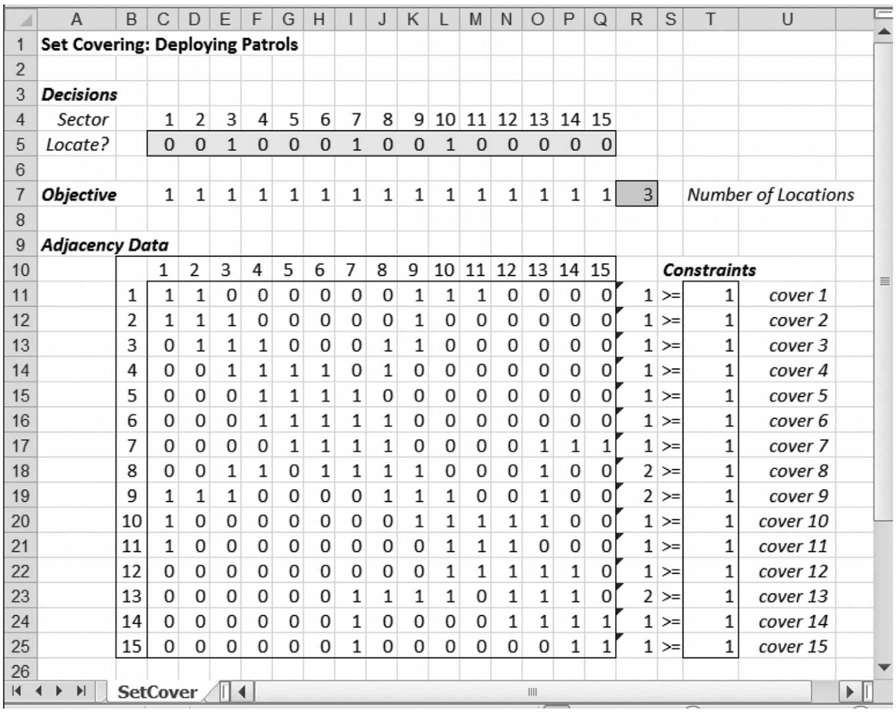
$$\sum_j a_{ij} y_j \geq 1, \quad (\text{for all sectors } i = 1, 2, \ldots, 15)$$

In the spreadsheet of Figure 6.11, we have positioned the constraint information immediately to the right of the data array because the rows of the array give rise to constraints. We then specify the model as follows.

| | |
|---|---|
| Objective: | R7 (minimize) |
| Variables: | C5:Q5 |
| Constraints: | R11:R25 $\geq$ T11:T25 |
| | C5:Q5 = binary |

Figure 6.11 shows the optimal solution, obtained with the linear solver and an Integer Tolerance of 0, which calls for assigning patrols to Sectors 3, 7, and 10. By using an integer programming model, the police department can provide service to Metropolis while minimizing the number of patrols needed.

The generic set covering problem involves the selection of objects to meet given coverage requirements, where selection is a matter of binary choice. Each column in the constraints of the model (refer to Figure 6.11 as an example) corresponds to an object and the coverage it provides. When the coverage is expressed with 0s and 1s (the constraint coefficients in a column), we can think of the 1s as defining a coverage



|  | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Set Covering: Deploying Patrols** | | | | | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | | | | |
| 3 | ***Decisions*** | | | | | | | | | | | | | | | | | | | | |
| 4 | *Sector* | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | | | |
| 5 | *Locate?* | | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | | | |
| 6 | | | | | | | | | | | | | | | | | | | | | |
| 7 | ***Objective*** | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | | *Number of Locations* | |
| 8 | | | | | | | | | | | | | | | | | | | | | |
| 9 | ***Adjacency Data*** | | | | | | | | | | | | | | | | | | | | |
| 10 | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | | ***Constraints*** | |
| 11 | | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | >= | 1 | *cover 1* |
| 12 | | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | >= | 1 | *cover 2* |
| 13 | | 3 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | >= | 1 | *cover 3* |
| 14 | | 4 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | >= | 1 | *cover 4* |
| 15 | | 5 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | >= | 1 | *cover 5* |
| 16 | | 6 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | >= | 1 | *cover 6* |
| 17 | | 7 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | >= | 1 | *cover 7* |
| 18 | | 8 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | >= | 1 | *cover 8* |
| 19 | | 9 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 2 | >= | 1 | *cover 9* |
| 20 | | 10 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | >= | 1 | *cover 10* |
| 21 | | 11 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | >= | 1 | *cover 11* |
| 22 | | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | >= | 1 | *cover 12* |
| 23 | | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 2 | >= | 1 | *cover 13* |
| 24 | | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | >= | 1 | *cover 14* |
| 25 | | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | >= | 1 | *cover 15* |
| 26 | | | | | | | | | | | | | | | | | | | | | |

**Figure 6.11.** Optimal solution for Example 6.3.

set (i.e., the set of rows in which 1s appear). Thus, the selection of objects is equivalent to the selection of sets, and the problem is to choose the minimum number of sets and still provide the required coverage. This interpretation gives rise to the name *set-covering problem.*

## 6.4. SET PACKING

As we saw in the previous section, the covering model of linear programming leads to an analog in binary-choice programming. In a similar fashion, the allocation model of linear program leads to another analog. This model contains allocation constraints rather than covering constraints; otherwise, it resembles the set-covering model. To describe how this model might arise, consider the expansion problem faced by the Happy Landings Motel Company.

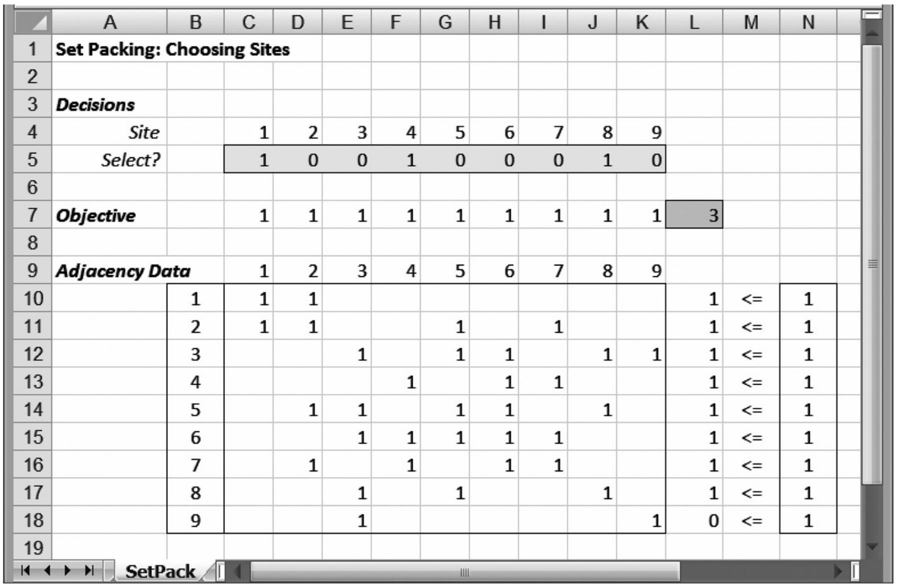
### EXAMPLE 6.4 *Happy Landings Motels*

The Happy Landings Motel has been a successful one-of-a-kind business in its original location. Its new owner has decided to replicate the motel at several locations and form a small chain. More expansion will follow if this first phase is a success. The new owner has identified nine potential locations where a motel might be located. However, some of these locations are within 30 miles of each other, and the owner prefers to separate motels in the chain by at least 30 miles to avoid cannibalizing demand. The following information about location conflicts has been obtained from a map.

| Potential site | Potential sites within 30 miles |
|---|---|
| 1 | 2 |
| 2 | 1, 5, 7 |
| 3 | 5, 6, 8, 9 |
| 4 | 6, 7 |
| 5 | 2, 3, 6, 8 |
| 6 | 3, 4, 5, 7 |
| 7 | 2, 4, 6 |
| 8 | 3, 5 |
| 9 | 3 |

The new owner wants to build motels on as many sites as possible, while adhering to the 30-mile requirement. ■

Again, the raw data for this problem comes from a map, organized according to the list given in Example 6.4. For modeling purposes, it's convenient to convert this information into a *conflict array*. This array is shown in Figure 6.12, as part of the ultimate model. In the figure, the numbered rows and columns correspond to the various



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Set Packing: Choosing Sites** | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | |
| 3 | ***Decisions*** | | | | | | | | | | | | | |
| 4 | *Site* | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | |
| 5 | *Select?* | | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | | | |
| 6 | | | | | | | | | | | | | | |
| 7 | ***Objective*** | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | | |
| 8 | | | | | | | | | | | | | | |
| 9 | ***Adjacency Data*** | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | |
| 10 | | 1 | 1 | 1 | | | | | | | | 1 | <= | 1 |
| 11 | | 2 | 1 | 1 | | | 1 | | 1 | | | 1 | <= | 1 |
| 12 | | 3 | | | 1 | | 1 | 1 | | 1 | 1 | 1 | <= | 1 |
| 13 | | 4 | | | | 1 | | 1 | 1 | | | 1 | <= | 1 |
| 14 | | 5 | | 1 | 1 | | 1 | 1 | | 1 | | 1 | <= | 1 |
| 15 | | 6 | | | 1 | 1 | 1 | 1 | 1 | | | 1 | <= | 1 |
| 16 | | 7 | | 1 | | 1 | | 1 | 1 | | | 1 | <= | 1 |
| 17 | | 8 | | | 1 | | 1 | | | 1 | | 1 | <= | 1 |
| 18 | | 9 | | | 1 | | | | | | 1 | 0 | <= | 1 |
| 19 | | | | | | | | | | | | | | |

**Figure 6.12.** Optimal solution for Example 6.4.

motel sites. An entry of 1 appears in row *i* and column *j* if sites *i* and *j* are within 30 miles of each other.

A model for maximizing the number of nonconflicting motel sites can be constructed around the conflict array by defining binary variables

$$y_j = 1 \quad \text{if a motel is assigned to site } j$$
$$y_j = 0 \quad \text{otherwise}$$

Then the objective function—the total number of motels—can be expressed as the sum of the $y_j$ values. In the model of Figure 6.12, this sum is represented as the SUMPRODUCT of the decision row and a row containing all 1s. To formulate constraints, suppose we focus on row *i* in the array, which corresponds to site *i*. The SUMPRODUCT of this row and the decisions yields the number of assigned sites (possibly including site *i* itself) that conflict with site *i*. In the problem, we require the number of conflicts to be at most 1, for any site. In words, our model takes the following form.

$$\text{Maximize } z = \text{ the number of sites}$$
$$\text{subject to}$$
$$\text{Sites in conflict with } i \leq 1, \quad (\text{for all } i = 1, 2, \ldots, 9)$$

In algebraic terms, we can let $a_{ij}$ represent the conflict data in the array of Figure 6.12. That is, $a_{ij} = 1$ if sites *i* and *j* are within 30 miles and $a_{ij} = 0$ if not. Then we can write

$$\text{Maximize } z = \sum y_j$$
$$\text{subject to}$$
$$\sum_j a_{ij} y_j \leq 1, \quad (\text{for all sites } i = 1, 2, \ldots, 9)$$

In the spreadsheet of Figure 6.12, we specify the model as follows.

| | |
|---|---|
| Objective: | L7 (maximize) |
| Variables: | C5:K5 |
| Constraints: | L10:L18 $\leq$ N10:N18 |
| | C5:K5 = binary |

Figure 6.12 shows the optimal solution, obtained with the linear solver and an Integer Tolerance of 0, which calls for choosing sites 1, 4, and 8. By using an integer programming model, the new owner can open as many new motels as the 30-mile limit allows.

The generic version of this problem involves the selection of objects that avoid given conflict possibilities, with selection represented by binary choice. Each column in the constraints of the model (refer to Figure 6.12 as an example) corresponds to an object and the conflicts it generates. When the conflicts are expressed with 0s and 1s (the constraint coefficients in a column), we can think of the 1s as defining a conflict set (i.e., the set of rows in which 1s appear.) Thus, the selection of objects is equivalent to the selection of sets, and the problem is to "pack in" the maximum number of sets without creating conflicts. This interpretation gives rise to the name *set-packing problem*.

## 6.5. SET PARTITIONING

As another example of a binary choice model, we consider a *matching problem*. The term "matching" refers to identifying pairs—that is, associating one object in a population with exactly one other object. This structure differs from the assignment problem of Chapter 3, which calls for matching an object in one population, such as products, to an object in another population, such as factories. Consider the exam-scheduling problem as it arises at Oxbridge College.

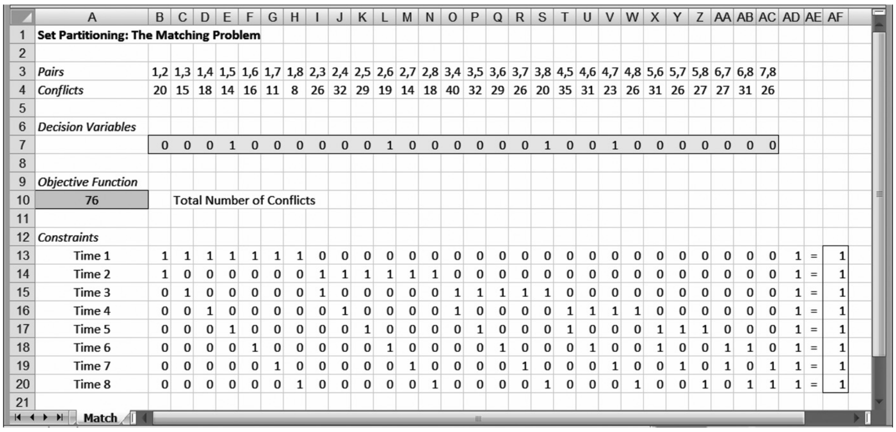
### EXAMPLE 6.5 *Oxbridge College*

Oxbridge College faces the problem of devising an exam schedule at the end of every term. By tradition, the exam period lasts four days, and exams are scheduled in the morning of each day. In other words, there are four available exam periods. To create an exam schedule, the college registrar assigns courses to exam days according to the course meeting time. Thus, all courses that meet Monday at 9am are assigned the same exam day. Eight distinct meeting times occur in the college calendar, and the registrar wants to assign two to each of the four exam days. However, when two times are assigned to the same exam day, some students may have a time conflict because they are enrolled in courses that meet at those two times. Special arrangements must then be made for such students. The registrar's office has an information system that can determine, for any pair of class times, how many students are taking courses at both times. With this information, the registrar would like to devise an exam schedule that makes the number of exam conflicts as small as possible because that minimizes the number of cases in which special arrangements have to be made. For the current term, the conflict numbers are displayed in the following table.

| Time | T2 | T3 | T4 | T5 | T6 | T7 | T8 |
|---|---|---|---|---|---|---|---|
| T1 | 20 | 15 | 18 | 14 | 16 | 11 | 8 |
| T2 | | 26 | 32 | 29 | 19 | 14 | 18 |
| T3 | | | 40 | 32 | 29 | 26 | 20 |
| T4 | | | | 35 | 31 | 23 | 26 |
| T5 | | | | | 31 | 26 | 27 |
| T6 | | | | | | 27 | 31 |
| T7 | | | | | | | 26 |

■

It's not necessary to fill in the entire array because of symmetry: the number of conflicts between T*i* and T*j* is the same as the number of conflicts between T*j* and T*i*.



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Set Partitioning: The Matching Problem** | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | |
| 3 | *Pairs* | 1,2 | 1,3 | 1,4 | 1,5 | 1,6 | 1,7 | 1,8 | 2,3 | 2,4 | 2,5 | 2,6 | 2,7 | 2,8 | 3,4 |
| 4 | *Conflicts* | 20 | 15 | 18 | 14 | 16 | 11 | 8 | 26 | 32 | 29 | 19 | 14 | 18 | 40 |
| 5 | | | | | | | | | | | | | | | |
| 6 | *Decision Variables* | | | | | | | | | | | | | | |
| 7 | | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 8 | | | | | | | | | | | | | | | |
| 9 | *Objective Function* | | | | | | | | | | | | | | |
| 10 | 76 | | Total Number of Conflicts | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | |
| 12 | *Constraints* | | | | | | | | | | | | | | |
| 13 | Time 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | Time 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 15 | Time 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 16 | Time 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 17 | Time 5 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 18 | Time 6 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 19 | Time 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 20 | Time 8 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 21 | | | | | | | | | | | | | | | |

| | A | P | Q | R | S | T | U | V | W | X | Y | Z | AA | AB | AC | AD | AE | AF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Set Partitioning: The Matching Problem** | | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | |
| 3 | *Pairs* | 3,5 | 3,6 | 3,7 | 3,8 | 4,5 | 4,6 | 4,7 | 4,8 | 5,6 | 5,7 | 5,8 | 6,7 | 6,8 | 7,8 | | | |
| 4 | *Conflicts* | 32 | 29 | 26 | 20 | 35 | 31 | 23 | 26 | 31 | 26 | 27 | 27 | 31 | 26 | | | |
| 5 | | | | | | | | | | | | | | | | | | |
| 6 | *Decision Variables* | | | | | | | | | | | | | | | | | |
| 7 | | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | |
| 8 | | | | | | | | | | | | | | | | | | |
| 9 | *Objective Function* | | | | | | | | | | | | | | | | | |
| 10 | 76 | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | |
| 12 | *Constraints* | | | | | | | | | | | | | | | | | |
| 13 | Time 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = | 1 |
| 14 | Time 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = | 1 |
| 15 | Time 3 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = | 1 |
| 16 | Time 4 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = | 1 |
| 17 | Time 5 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | = | 1 |
| 18 | Time 6 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | = | 1 |
| 19 | Time 7 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | = | 1 |
| 20 | Time 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | = | 1 |
| 21 | | | | | | | | | | | | | | | | | | |

**Figure 6.13.** Spreadsheet model for Example 6.5.

Although it is convenient to present the data in a half-array, it can also be presented in a horizontal layout, as in the standard linear programming format. (See rows 3 and 4 of Figure 6.13.)

The essential decision in this problem is a binary choice for matching. The binary decision variables $x_{ij}$ are defined as follows.

$x_{ij} = 1$ if class time T$i$ and class time T$j$ are assigned to the same exam day.
$x_{ij} = 0$ otherwise

Because symmetry allows us to work with just half the conflict array, we need to define the $x_{ij}$ variables only for $i < j$, which comes to 28 pairs. Then we can write the objective function as a SUMPRODUCT in the form $\sum c_{ij}x_{ij}$, where $c_{ij}$ represents the number of conflicts occurring when T$i$ and T$j$ are assigned to the same exam day, and where the sum is taken over all 28 potential assignments. Because the quantity $c_{ij}$ contributes to this sum only when $x_{ij} = 1$, the objective function measures the total number of conflicts in the exam schedule. Constraints are needed to make sure that each class time is assigned exactly one match. The algebraic model takes the following form.

$$\text{Minimize } z = 20x_{12} + 15x_{13} + 18x_{14} + \quad \cdots \quad + 26x_{78}$$

subject to

$$\begin{aligned}
x_{12} &+ x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} = 1\\
x_{12} &+ x_{23} + x_{24} + x_{25} + x_{26} + x_{27} + x_{28} = 1\\
x_{13} &+ x_{23} + x_{34} + x_{35} + x_{36} + x_{37} + x_{38} = 1\\
x_{14} &+ x_{24} + x_{34} + x_{45} + x_{46} + x_{47} + x_{48} = 1\\
x_{15} &+ x_{25} + x_{35} + x_{45} + x_{56} + x_{57} + x_{58} = 1\\
x_{16} &+ x_{26} + x_{36} + x_{46} + x_{56} + x_{67} + x_{68} = 1\\
x_{17} &+ x_{27} + x_{37} + x_{47} + x_{57} + x_{67} + x_{78} = 1\\
x_{18} &+ x_{28} + x_{38} + x_{48} + x_{58} + x_{68} + x_{78} = 1
\end{aligned}$$

Figure 6.13 shows the spreadsheet model. The layout follows the standard linear programming layout introduced in Chapter 2. By transforming the layout of the conflict data from the original half-array into one long row in Figure 6.13, we facilitate the standard layout.

The model specification is the following.

| | |
|---|---|
| Objective: | A10 (minimize) |
| Variables: | B7:AC7 |
| Constraints: | AD13:AD20 = 1 |
| | B7:AC7 = binary |

After entering this information, we set Integer Tolerance to 0 and run the linear solver. The minimal number of conflicts is 76, as shown in the figure, and the optimal pairings are Time 1 with Time 5, Time 2 with Time 6, Time 3 with Time 8, and Time 4 with Time 7. By using an integer programming model, Oxbridge can thus limit the number of exam conflicts to the minimum possible level.

The binary-choice model in Figure 6.13 resembles the set covering model and the set packing model, except for the constraint type. Again, we can think of columns as sets, and in this case, the 1s in each column identify the pair of objects in the corresponding match. The problem involves choosing a full match—that is, a collection of pairs such that each object appears exactly once in the collection. The "exactly once" requirement means that the sets selected must partition the population of objects into mutually exclusive and exhaustive subsets. For that reason, this problem type is often called a *set-partitioning problem*.

In Example 6.5, the number of class times is exactly equal to twice the number of exam days. More generally, the number of class times could be slightly less than twice the number of exam days, and a similar model could be used. For example, if there were six class times rather than eight, then only two pairings would be needed, and two of the class times would be unpaired. To accommodate this condition, we could state the model constraints as LT inequalities and add a constraint to ensure that at least two pairings took place.

Although Example 6.5 occurs in the setting of scheduling exams, the same kind of matching model can be used to schedule class meetings (or training courses, or conference sessions, etc.) when there are at most two items per time period. Several additional conditions may apply as well, but the basic structure of such problems often corresponds to the matching problem.

## 6.6. PLAYOFF SCHEDULING

An application area for integer programming that has received increasing attention lately is the scheduling of sports teams in professional leagues. Although basic guidelines exist for the creation of a "balanced" schedule, a good deal of flexibility remains, and several specific considerations come into play in determining an "optimal" schedule. As an example, a relatively new league in Latin America has just begun to examine the possibility of optimized scheduling.