# Chapter 3
# Mixed-Integer Linear Optimization

In this chapter, we study mixed-integer linear optimization problems. Also known as mixed-integer linear programming problems (MILPPs), these are problems with an objective function and constraints that are all linear in the decision variables. What sets MILPPs apart from linear programming problems (LPPs) is that at least some of the variables in MILPPs are constrained to take on integer values. LPPs, conversely, have no such constraints and all of the variables can take on any continuous value.

This chapter begins by providing a number of illustrative examples to show the practical significance of MILPPs. Then, a general formulation of the MILPP is provided. Next, we demonstrate the use of a special type of integer variable known as a binary variable. We show that binary variables can be used to model a number of types of nonlinearities and discontinuities while maintaining a linear model structure. This use of binary variables is, in some sense, the true power of mixed-integer linear optimization models. Two solution techniques for MILPPs are then introduced. We first discuss the use of a branch-and-bound method to solve general MILPPs. We next introduce a cutting-plane algorithm for MILPPs in which *all* of the variables are constrained to take on integer values. Problems with this structure are commonly referred to as pure-integer linear optimization problems or pure-integer linear programming problems (PILPPs). This chapter closes with some final remarks, GAMS codes for the illustrative examples, and a number of end-of-chapter exercises.

## 3.1 Motivating Examples

This introductory section provides and explains a number of illustrative examples to show the practical value of MILPPs. These examples pertain to the energy sector and cover most of the standard 'classes' of MILPPs.



R. Sioshansi and A.J. Conejo, *Optimization in Engineering*,
Springer Optimization and Its Applications 120, DOI 10.1007/978-3-319-56769-3_3

### 3.1.1 Photovoltaic Panel-Repair Problem

A spacecraft has a series of photovoltaic (PV) solar panels installed that must be repaired. There are two types of repair units, type A and B, that can be used to repair the PV panels. One type-A unit has a mass of 17 kg and occupies 32 $m^3$ of space while one type-B unit has a mass of 32 kg and occupies 15 $m^3$. The shuttle that is available to transport repair units to the spacecraft can only accommodate up to 136 kg and up to 120 $m^3$. How many units of each type (A and B) should be transported to maximize the total number of repair units sent to the spacecraft?

There are two decision variables in this problem. We let $x_1$ denote the number of type-A units transported to the spacecraft and $x_2$ the number of type-B units transported.

The objective of this problem is to maximize the total number of repair units transported:

$$\max_{x_1,x_2} \; x_1 + x_2.$$

There are four sets of constraints in this problem. First, we must impose the mass constraint, which is:

$$17x_1 + 32x_2 \leq 136.$$

Second, we have the volume constraint:

$$32x_1 + 15x_2 \leq 120.$$

Third, we must constrain each of $x_1$ and $x_2$ to be non-negative, because transporting a negative number of repair units is physically meaningless:

$$x_1, x_2 \geq 0.$$

Finally, we impose a constraint that each of the variables, $x_1$ and $x_2$, must take on integer values. The most compact way to express this is:

$$x_1, x_2 \in \mathbb{Z},$$

as $\mathbb{Z}$ is the standard notation for the set of integers.

When we impose this fourth constraint, the two variables $x_1$ and $x_2$ become **integer variables**. This should be contrasted with all of the variables used in formulating LPPs in Chapter 2. We have no restriction in any of the models formulated in Chapter 2 that the variables take on integer values. Indeed, the Simplex method, which is used to solve LPPs, has no general guarantee that any of the optimal decision variable values obtained have integer values. It is straightforward to verify that one can formulate a multitude of LPPs that do not yield integer-valued decision variables.

Taking all of these elements together, the problem can be formulated as:

$$\max_{x_1,x_2} z = x_1 + x_2 \tag{3.1}$$
$$\text{s.t. } 17x_1 + 32x_2 \leq 136 \tag{3.2}$$
$$32x_1 + 15x_2 \leq 120 \tag{3.3}$$
$$x_1, x_2 \geq 0 \tag{3.4}$$
$$x_1, x_2 \in \mathbb{Z}. \tag{3.5}$$

Because problem (3.1)–(3.5) includes integer variables, we refer to it as a **mixed-integer linear optimization problem**. Indeed, because all of the variables in this particular problem are restricted to take on integer values, we can refer to it more specifically as a **pure-integer linear optimization problem**. The distinction between a mixed- and pure-integer optimization problem is that the former can include a mix of variables that are and are not restricted to take on integer values. The latter *only* includes variables restricted to take on integer values.

Before proceeding, we finally note that we can express this optimization problem in the even more compact form:

$$\begin{aligned} \max_{x_1,x_2} z &= x_1 + x_2 \\ \text{s.t. } 17x_1 + 32x_2 &\leq 136 \\ 32x_1 + 15x_2 &\leq 120 \\ x_1, x_2 &\in \mathbb{Z}^+, \end{aligned}$$

where $\mathbb{Z}^+$ is the standard notation for the set of non-negative integers.

The feasible region of problem (3.1)–(3.5) is shown in Figure 3.1. All of the feasible (integer) solutions are indicated in this figure by small circles and we see

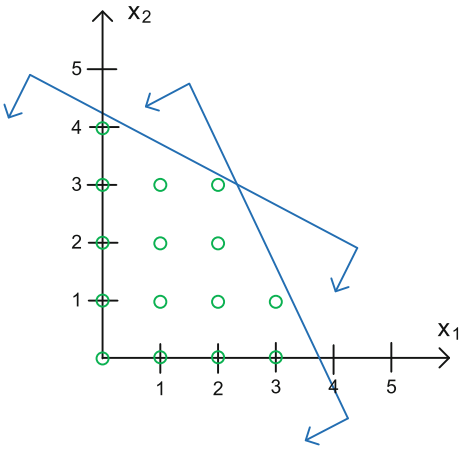


**Fig. 3.1** Geometrical representation of the feasible region of the Photovoltaic Panel-Repair Problem

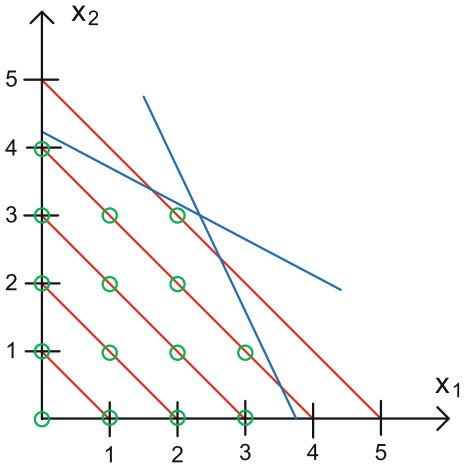


**Fig. 3.2** Geometrical representation of the feasible region of the Photovoltaic Panel-Repair Problem with objective-function contour plot overlaid

that there are a total of 15 feasible points: $(x_1, x_2)$ can feasibly equal any of $(0, 0)$, $(0, 1)$, $(0, 2)$, $(0, 3)$, $(0, 4)$, $(1, 0)$, $(1, 1)$, $(1, 2)$, $(1, 3)$, $(2, 0)$, $(2, 1)$, $(2, 2)$, $(2, 3)$, $(3, 0)$, or $(3, 1)$.

Figure 3.2 overlays the contour plot of the objective function on the feasible region. Visual inspection of this figure shows that of the 15 feasible solutions, $(x_1, x_2) = (2, 3)$ is optimal because it allows the most (five) repair units to be sent to the spacecraft.

We finally note that the Photovoltaic Panel-Repair Problem is a simplified instance of what is known as a knapsack problem.

### *3.1.2 Natural Gas-Storage Problem*

A natural gas supplier needs to build gas-storage facilities with which to supply customers in two cities. The company has three possible sites where storage facilities can be built. Table 3.1 indicates how much revenue is earned (in $ billion) by the company per GJ of natural gas supplied from each of the three prospective facilities to each of the two cities.

Table 3.2 summarizes the building cost (in $ billion) of each of the three prospective gas-storage facilities. It also lists the capacity of each of the facilities, in GJ, if that facility is built. Note that each of the costs is incurred if the corresponding gas-storage facility is built, irrespective of how much gas is ultimately stored there.

**Table 3.1** Revenue earned from each potential gas-storage facility [$ billion/GJ] in the Natural Gas-Storage Problem

| | City 1 | City 2 |
|---|---|---|
| Storage Facility 1 | 1 | 6 |
| Storage Facility 2 | 2 | 5 |
| Storage Facility 3 | 3 | 4 |

**Table 3.2** Building cost and capacity of gas-storage facilities in the Natural Gas-Storage Problem

| Storage Facility | Cost [$ billion] | Capacity [GJ] |
|---|---|---|
| 1 | 8 | 7 |
| 2 | 9 | 8 |
| 3 | 7 | 9 |

Finally, Table 3.3 summarizes the demand for natural gas (in GJ) in each city, which must be exactly met. The company would like to determine which (if any) of the three gas-storage facilities to build and how much gas to supply from each facility built to each city to maximize its profit.

**Table 3.3** Demand for natural gas in each city in the Natural Gas-Storage Problem

| City | Demand [GJ] |
|---|---|
| 1 | 10 |
| 2 | 6 |

To formulate this problem, we introduce two sets of variables. The first, which we denote $x_1$, $x_2$, and $x_3$, represent the company's decision of whether to build each of the three storage facilities. We model these using a specific type of integer variable, called a **binary variable**. As the name suggests, a binary variable is restricted to take on two values, which are almost always 0 and 1. Binary variables are almost always used to model logical conditions or decisions. In this example, we define each of the three $x$ variables to represent the logical decision of whether to build each of the three storage facilities. Thus, we define each of these variables as:

$$x_1 = \begin{cases} 1, \text{ if gas-storage facility 1 is built,} \\ 0, \text{ if gas-storage facility 1 is not built,} \end{cases}$$

$$x_2 = \begin{cases} 1, \text{ if gas-storage facility 2 is built,} \\ 0, \text{ if gas-storage facility 2 is not built,} \end{cases}$$

and:

$$x_3 = \begin{cases} 1, \text{ if gas-storage facility 3 is built,} \\ 0, \text{ if gas-storage facility 3 is not built.} \end{cases}$$

We next define an additional set of six variables, which we denote $y_{i,j}$ where $i = 1, 2, 3$ and $j = 1, 2$. The variable $y_{i,j}$ is defined as the amount of natural gas (in GJ) supplied by storage facility $i$ to city $j$.

The company's objective is to maximize its profit, which is defined as revenue less cost. Using the data reported in Table 3.1, the company's revenue is defined as:

$$1y_{1,1} + 2y_{2,1} + 3y_{3,1} + 6y_{1,2} + 5y_{2,2} + 4y_{3,2}.$$

The company incurs costs for building gas-storage facilities, which are summarized in Table 3.2. Indeed, we can compute the company's cost by multiplying the cost data reported in Table 3.2 by each of the corresponding $x$ variables. This gives:

$$8x_1 + 9x_2 + 7x_3,$$

as the company's cost. To understand this expression, note that if gas-storage facility $i$ is not built, then $x_i = 0$. In that case, the product of $x_i$ and the corresponding cost will be zero, meaning that the company incurs no cost for building storage facility $i$. If, on the other hand, facility $i$ is built, then $x_i = 1$ and the product of $x_i$ and the corresponding cost gives the correct cost. Taking the revenue and cost expressions together, the company's objective function is:

$$\max_{x,y} \; y_{1,1} + 2y_{2,1} + 3y_{3,1} + 6y_{1,2} + 5y_{2,2} + 4y_{3,2} - (8x_1 + 9x_2 + 7x_3).$$

This problem has four types of constraints. The first imposes the requirement that the demand in each city is met:

$$y_{1,1} + y_{2,1} + y_{3,1} = 10,$$

and:

$$y_{1,2} + y_{2,2} + y_{3,2} = 6.$$

We next need constraints that ensure that none of the three gas-storage facilities operate above maximum capacities. One may be tempted to write these constraints as:

$$y_{1,1} + y_{1,2} \leq 7,$$

$$y_{2,1} + y_{2,2} \leq 8,$$

and:

$$y_{3,1} + y_{3,2} \leq 9.$$

However, it is actually preferable to write these constraints as:

$$y_{1,1} + y_{1,2} \leq 7x_1,$$

$$y_{2,1} + y_{2,2} \leq 8x_2,$$

and:

$$y_{3,1} + y_{3,2} \leq 9x_3.$$

The reason for this is that the second set of constraints, with the $x$'s on the right-hand sides, impose an additional restriction that a gas-storage facility cannot operate (meaning that its maximum capacity is zero) if the facility is not built. To understand this, let us examine the first of the three constraints:

$$y_{1,1} + y_{1,2} \leq 7x_1.$$

If gas-storage facility 1 is built, then $x_1 = 1$ and this constraint simplifies to:

$$y_{1,1} + y_{1,2} \leq 7,$$

which is what we want (the gas-storage facility can hold at most 7 GJ of gas). Otherwise, if the facility 1 is not built, then $x_1 = 0$ and the constraint simplifies to:

$$y_{1,1} + y_{1,2} \leq 0.$$

This is, again, what we want in this case because if facility 1 is *not* built, then it can hold 0 GJ of gas. These three constraints are an example of **logical constraints**. This is because they encode a logical condition. In this case, a gas-storage facility only has capacity available if it is built. Otherwise, it has zero capacity available. We next require the $y$ variables to be non-negative:

$$y_{1,1}, y_{1,2}, y_{2,1}, y_{2,2}, y_{3,1}, y_{3,2} \geq 0.$$

We finally require the $x$ variables to be binary:

$$x_1, x_2, x_3 \in \{0, 1\}.$$

Taking all of these elements together, the MILPP for this problem is:

$$\begin{aligned}
\max_{x,y}\ & z = y_{1,1} + 2y_{2,1} + 3y_{3,1} + 6y_{1,2} + 5y_{2,2} + 4y_{3,2} - 8x_1 - 9x_2 - 7x_3 \\
\text{s.t. } & y_{1,1} + y_{2,1} + y_{3,1} = 10 \\
& y_{1,2} + y_{2,2} + y_{3,2} = 6 \\
& y_{1,1} + y_{1,2} \leq 7x_1 && (3.6) \\
& y_{2,1} + y_{2,2} \leq 8x_2 && (3.7) \\
& y_{3,1} + y_{3,2} \leq 9x_3 && (3.8) \\
& y_{i,j} \geq 0, \forall\, i = 1, 2, 3;\ j = 1, 2 \\
& x_i \in \{0, 1\}, \forall\, i = 1, 2, 3.
\end{aligned}$$

The Natural Gas-Storage Problem is a simplified version of a facility-location problem.

### 3.1.3 *Electricity-Scheduling Problem*

An electricity producer needs to supply 50 kW of power to a remote village over the next hour. To do so, it has three generating units available. The producer must decide whether to switch each of the three generating units on or not.

If it does switch a generating unit on, the producer must pay a fixed cost to operate the unit, which is independent of how much energy the unit produces. In addition to this fixed cost, there is a variable cost that the producer must pay for each kWh produced by each unit.

If a unit is switched on, it must produce any amount of energy between a minimum and maximum production level. Otherwise, its production level must equal zero. Table 3.4 summarizes the operating costs and production limits of the electricity-production units. The electricity producer would like to schedule the three production units to minimize the cost of exactly serving the 50 kW demand of the village.

**Table 3.4** Operating costs and production limits of electricity-production units in the Electricity-Scheduling Problem

| Production Unit | Cost | | Production Limits [kW] | |
|---|---|---|---|---|
| | Variable [$/kWh] | Fixed [$/hour] | Minimum | Maximum |
| 1 | 2 | 40 | 5 | 20 |
| 2 | 5 | 50 | 6 | 40 |
| 3 | 1 | 35 | 4 | 35 |

To formulate this problem, we introduce two sets of variables. We first have variables $p_1$, $p_2$ and $p_3$, where we define $p_i$ as the kW produced by unit $i$. We next define three binary variables, $x_1$, $x_2$, and $x_3$, where we define $x_i$ as:

$$x_i = \begin{cases} 1, & \text{if unit } i \text{ is switched on,} \\ 0, & \text{if unit } i \text{ is not switched on.} \end{cases}$$

The producer's objective function, which is to minimize cost, can be written as:

$$\min_{p,x} \; 2p_1 + 5p_2 + 1p_3 + 40x_1 + 50x_2 + 35x_3.$$

The problem requires three types of constraints. We first need a constraint that ensures that the 50 kW of demand is exactly met:

$$p_1 + p_2 + p_3 = 50.$$

We next must impose the production limits of the units, which can be written as:

$$5x_1 \leq p_1 \leq 20x_1,$$

$$6x_2 \leq p_2 \leq 40x_2,$$

and:

$$4x_3 \leq p_3 \leq 35x_3.$$

Note that these three production-limit constraints have the logical constraint involving the decision to switch each unit on or not embedded within them. To see this, let us look more closely at the first of these three constraints:

$$5x_1 \leq p_1 \leq 20x_1.$$

If unit 1 is switched on, then $x_1 = 1$ and this constraint simplifies to:

$$5 \leq p_1 \leq 20.$$

This is the constraint that we want in this case, because if unit 1 is switched on, then it must produce between its minimum (5 kW) and maximum (20 kW) production levels. Otherwise, if unit 1 is not switched on then $x_1 = 0$ and the production-limit constraint becomes:

$$0 \leq p_1 \leq 0,$$

or simply:

$$p_1 = 0.$$

This is, again, correct because if unit 1 is not switched on, its production must be equal to zero. We finally have a constraint that the $x$ variables be binary:

$$x_1, x_2, x_3 \in \{0, 1\}.$$

Thus, the MILPP for this problem is:

$$\begin{aligned}
&\min_{p,x} z = 2p_1 + 5p_2 + p_3 + 40x_1 + 50x_2 + 35x_3 \\
&\text{s.t. } p_1 + p_2 + p_3 = 50 \\
&\quad 5x_1 \leq p_1 \leq 20x_1 && (3.9) \\
&\quad 6x_2 \leq p_2 \leq 40x_2 && (3.10) \\
&\quad 4x_3 \leq p_3 \leq 35x_3 && (3.11) \\
&\quad x_i \in \{0, 1\}, \forall\, i = 1, 2, 3.
\end{aligned}$$

Although it is physically meaningless for any of the $p_1$, $p_2$, or $p_3$ to take on a negative values, we do not need to include explicit non-negativity constraints for these variables. This is because the left-hand sides of double-sided inequalities (3.9)–(3.11) ensure that each of the $p$ variables take on non-negative values. We could include non-negativity constraints for the $p$ variables, however, they would be redundant in this problem.

We finally note that this example is a simplified instance of a unit-scheduling problem.

### 3.1.4 Oil-Transmission Problem

An oil company owns two oil wells that are connected via three pipelines to one another and to a refinery, as shown in Figure 3.3. The refinery has a fixed demand for 30 t of oil. The company is considering expanding the capacity of the two pipelines that directly connect the wells with the refinery, as depicted in the figure.

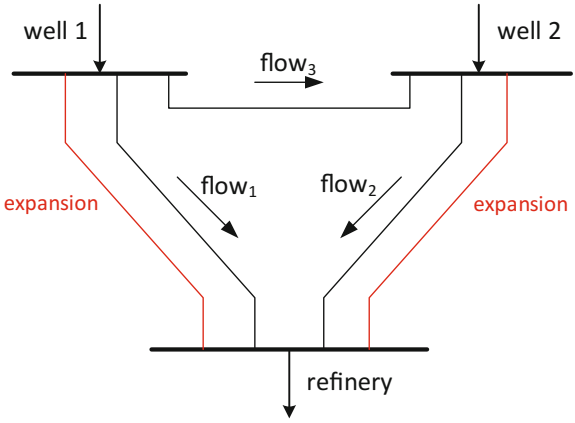


**Fig. 3.3** Network in the Oil-Transmission Problem

The company earns a profit of \$2000 per ton of oil sold from well 1 to the refinery and a profit of \$3000/t for oil sold from well 2 to the refinery.

The pipeline that directly connects well 1 with the refinery can carry 12 t of oil. If it is expanded, which costs \$50000, then this capacity increases by 11 t to 23 t total. Similarly, the pipeline directly connecting well 2 with the refinery can carry 11 t of oil. This capacity increases by 12 t if the pipeline is expanded, which costs \$55000 to do. The pipeline directly connecting wells 1 and 2 can carry at most 10 t and this pipeline *cannot* be expanded.

The company would like to determine which (if any) of the pipelines to expand and how to ship oil from its wells to the refinery to maximize its profit.

To formulate this problem, we introduce three sets of variables. First, we let $p_1$ and $p_2$ denote the amount of oil extracted from each of wells 1 and 2, respectively. We next define $f_1$, $f_2$, and $f_3$ as the flows along the three pipelines, as shown in Figure 3.3. We use the convention that if $f_i$ is positive, that means there is a net flow in the direction of the arrow depicted in the figure. We finally define two binary variables, $x_1$ and $x_2$, representing the pipeline-expansion decisions. More specifically, we define $x_i$ as:

$$x_i = \begin{cases} 1, & \text{if pipeline } i \text{ is expanded,} \\ 0, & \text{if pipeline } i \text{ is not expanded.} \end{cases}$$

The company's objective function, which maximizes profit, is:

$$\max_{p,f,x} \; 2000p_1 + 3000p_2 - 50000x_1 - 55000x_2.$$

This problem has five types of constraints. First, we must ensure that we deliver exactly 30 t of oil to the refinery:

$$p_1 + p_2 = 30.$$

This constraint ensures that a total of exactly 30 t of oil is extracted from the two wells. Next, we must ensure that the oil extracted at the two wells is not left at the two wells but is instead delivered to the refinery through the pipeline network. We write these constraints as:

$$p_1 = f_1 + f_3,$$

and:

$$p_2 = f_2 - f_3.$$

These two constraints require that the total amount of oil extracted at each well (which is on the left-hand sides of the constraints) exactly equals the amount of oil that leaves each well through the pipelines (which is on the right-hand sides of the constraints). We next impose the flow constraints on the three pipelines:

$$-12 - 11x_1 \leq f_1 \leq 12 + 11x_1,$$

$$-11 - 12x_2 \leq f_2 \leq 11 + 12x_2,$$

and:

$$-10 \leq f_3 \leq 10.$$

We next require that the amount produced by each well be non-negative:

$$p_1, p_2 \geq 0,$$

and that the $x$ variables be binary:

$$x_1, x_2 \in \{0, 1\}.$$

Putting these elements together, this problem is formulated as:

$$\begin{aligned}
\max_{p,f,x} \; & z = 2000p_1 + 3000p_2 - 50000x_1 - 55000x_2 \\
\text{s.t. } & p_1 + p_2 = 30 \\
& p_1 = f_1 + f_3 \\
& p_2 = f_2 - f_3 \\
& -12 - 11x_1 \leq f_1 \leq 12 + 11x_1 \\
& -11 - 12x_2 \leq f_2 \leq 11 + 12x_2 \\
& -10 \leq f_3 \leq 10 \\
& p_i \geq 0, \forall\, i = 1, 2 \\
& x_i \in \{0, 1\}, \forall\, i = 1, 2.
\end{aligned}$$

This problem is a simple version of a transmission-expansion problem.

### *3.1.5 Charging-Station Problem*

To serve electric vehicle (EV) owners in four neighborhoods, a city needs to identify which (if any) of three potential EV charging stations to build. The city's goal is to minimize the total cost of building the stations, while properly serving the EV-charging needs of the four neighborhoods.

Table 3.5 summarizes which of the four neighborhoods can be served by each of the three potential EV-charging-station locations. An entry of 1 in the table means that the neighborhood can be served by a station at the location while an entry of 0 means that it cannot be. Table 3.6 summarizes the cost incurred for building each of the three potential EV charging stations.

**Table 3.5** Neighborhoods that can use each potential EV-charging-station location in the Charging-Station Problem

| | Location 1 | Location 2 | Location 3 |
|---|---|---|---|
| Neighborhood 1 | 1 | 0 | 1 |
| Neighborhood 2 | 0 | 1 | 0 |
| Neighborhood 3 | 1 | 1 | 0 |
| Neighborhood 4 | 0 | 0 | 1 |

**Table 3.6** Cost of building EV-charging stations in the Charging-Station Problem

| | Cost [$ million] |
|---|---|
| Location 1 | 10 |
| Location 2 | 12 |
| Location 3 | 13 |

To formulate this problem, we define three binary variables, $x_1$, $x_2$ and $x_3$, where we define $x_i$ as:

$$x_i = \begin{cases} 1, \text{ if EV-charging station } i \text{ is built,} \\ 0, \text{ if EV-charging station } i \text{ is not built.} \end{cases}$$

The city's objective function is to minimize cost, which is given by:

$$\min_x \ 10x_1 + 12x_2 + 13x_3,$$

where the objective is measured in millions of dollars.

There are two types of constraints in this problem. First, we must ensure that the EV charging stations built are capable of serving EV owners in each of the four neighborhoods. These constraints take the form:

$$1x_1 + 0x_2 + 1x_3 \geq 1,$$

$$0x_1 + 1x_2 + 0x_3 \geq 1,$$

$$1x_1 + 1x_2 + 0x_3 \geq 1,$$

and:

$$0x_1 + 0x_2 + 1x_3 \geq 1.$$

To understand the logic behind each of these constraints, let us examine the first one:

$$1x_1 + 0x_2 + 1x_3 \geq 1,$$

more closely. We know, from Table 3.5 that EV owners in neighborhood 1 can only be served by stations 1 or 3. Thus, at least one of those two stations must be built. The constraint imposes this requirement by multiplying each of $x_1$ and $x_3$ by 1. The sum:

$$1x_1 + 0x_2 + 1x_3 = x_1 + x_3,$$

measures how many of the stations that can serve EV owners in neighborhood 1 are built. The constraint requires that at least one such station be built. An analysis of the other three constraints have the same interpretation. We must also impose a constraint that the $x$ variables be binary:

$$x_1, x_2, x_3 \in \{0, 1\}.$$

Taking these together, the problem is formulated as:

$$\begin{aligned} \min_x \; & z = 10x_1 + 12x_2 + 13x_3 \\ \text{s.t. } & x_1 + x_3 \geq 1 \\ & x_2 \geq 1 \\ & x_1 + x_2 \geq 1 \\ & x_3 \geq 1 \\ & x_i \in \{0, 1\}, \forall\, i = 1, 2, 3. \end{aligned}$$

The Charging-Station Problem is an example of an area-covering or set-covering problem.

### *3.1.6 Wind Farm-Maintenance Problem*

A wind-generation company must perform annual maintenance on its three wind farms. The company has three maintenance teams to carry out this work. Each maintenance team must be assigned to exactly one wind farm and each wind farm must have exactly one maintenance team assigned to it. Table 3.7 lists the costs of assigning the three maintenance teams to each wind farm. The company would like to determine the assignments to minimize its total maintenance costs.

**Table 3.7** Cost of assigning maintenance crews to wind farms in the Wind Farm-Maintenance Problem

| | Wind Farm 1 | Wind Farm 2 | Wind Farm 3 |
|---|---|---|---|
| Maintenance Team 1 | 10 | 12 | 14 |
| Maintenance Team 2 | 9 | 8 | 15 |
| Maintenance Team 3 | 10 | 5 | 15 |

To formulate this problem, we define one set of variables, which we denote $x_{i,j}$ with $i = 1, 2, 3$ and $j = 1, 2, 3$, where we define $x_{i,j}$ as:

$$x_{i,j} = \begin{cases} 1, \text{ if maintenance crew } i \text{ is assigned to wind farm } j, \\ 0, \text{ otherwise.} \end{cases}$$

The company's objective function is to minimize cost, which is given by:

$$\min_x \; 10x_{1,1} + 12x_{1,2} + 14x_{1,3} + 9x_{2,1} + 8x_{2,2} + 15x_{2,3} + 10x_{3,1} + 5x_{3,2} + 15x_{3,3}.$$

This problem has three sets of constraints. First, we must ensure that each maintenance crew is assigned to exactly one wind farm, giving the following constraints:

$$x_{1,1} + x_{1,2} + x_{1,3} = 1,$$

$$x_{2,1} + x_{2,2} + x_{2,3} = 1,$$

and:

$$x_{3,1} + x_{3,2} + x_{3,3} = 1.$$

We must next ensure that each wind farm has exactly one maintenance crew assigned to it:

$$x_{1,1} + x_{2,1} + x_{3,1} = 1,$$

$$x_{1,2} + x_{2,2} + x_{3,2} = 1,$$

and:

$$x_{1,3} + x_{2,3} + x_{3,3} = 1.$$

We must finally ensure that all of the variables are binary:

$$x_{i,j} \in \{0, 1\}, \forall\, i = 1, 2, 3;\ j = 1, 2, 3.$$

Taking these elements together, the company's problem is:

$$\begin{aligned}
\max_{x}\ & z = 10x_{1,1} + 12x_{1,2} + 14x_{1,3} + 9x_{2,1} + 8x_{2,2} + 15x_{2,3} \\
& \quad + 10x_{3,1} + 5x_{3,2} + 15x_{3,3} && (3.12)\\
\text{s.t. } & x_{1,1} + x_{1,2} + x_{1,3} = 1 && (3.13)\\
& x_{2,1} + x_{2,2} + x_{2,3} = 1 && (3.14)\\
& x_{3,1} + x_{3,2} + x_{3,3} = 1 && (3.15)\\
& x_{1,1} + x_{2,1} + x_{3,1} = 1 && (3.16)\\
& x_{1,2} + x_{2,2} + x_{3,2} = 1 && (3.17)\\
& x_{1,3} + x_{2,3} + x_{3,3} = 1 && (3.18)\\
& x_{i,j} \in \{0, 1\}, \forall\, i = 1, 2, 3;\ j = 1, 2, 3. && (3.19)
\end{aligned}$$

This problem is an example of an assignment problem.

## 3.2 Types of Mixed-Integer Linear Optimization Problems

Mixed-integer linear optimization problems can take a number of different more specific forms. We have discussed some examples of these, such as pure-integer linear optimization problems. We describe some of these forms in more detail here.

### *3.2.1 General Mixed-Integer Linear Optimization Problems*

A general mixed-integer linear optimization problem or MILPP has the form:

$$\begin{aligned}
\min_{x_1,\ldots,x_n} \quad & c_0 + \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} A^e_{j,i} x_i = b^e_j, && \forall\ j = 1, \ldots, m_e \\
& \sum_{i=1}^{n} A^g_{j,i} x_i \geq b^g_j, && \forall\ j = 1, \ldots, m_g \\
& \sum_{i=1}^{n} A^l_{j,i} x_i \leq b^l_j, && \forall\ j = 1, \ldots, m_l \\
& x_i \in \mathbb{Z}, && \text{for some } i = 1, \ldots, n \\
& x_i \in \mathbb{R}, && \text{for the remaining } i = 1, \ldots, n,
\end{aligned}$$

where $m_e$, $m_g$, and $m_l$ are the numbers of equal-to, greater-than-or-equal-to, and less-than-or-equal to constraints. Thus, $m = m_e + m_g + m_l$ is the total number of constraints. The coefficients, $A^e_{j,i}, \forall i = 1, \ldots, n,\ j = 1, \ldots, m_e$, $A^g_{j,i}, \forall i = 1, \ldots, n,\ j = 1, \ldots, m_g$, and $A^l_{j,i}, \forall i = 1, \ldots, n,\ j = 1, \ldots, m_l$, the terms on the right-hand sides of the constraints, $b^e_j, \forall j = 1, \ldots, m_e$, $b^g_j, \forall j = 1, \ldots, m_g$, and $b^l_j, \forall j = 1, \ldots, m_l$, and the coefficients, $c_0, \ldots, c_n$, in the objective function are all constants.

Some subset of the variables are restricted to take on integer values:

$$x_i \in \mathbb{Z}, \text{for some } i = 1, \ldots, n,$$

where:

$$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\},$$

is standard notation for the set of integers. These are called the integer variables. The remaining variables have no such restriction and can take on any non-integer values that satisfy the remaining constraints.

Of the examples given in Section 3.1, the Natural Gas-Storage, Electricity-Scheduling and Oil-Transmission Problems, which are introduced in Sections 3.1.2, 3.1.3 and 3.1.4, respectively, are general MILPPs.

### 3.2.2 Pure-Integer Linear Optimization Problems

A pure-integer linear optimization problem or PILPP is a special case of a general MILPP in which *all* of the variables are restricted to take on integer values. A PILPP has the generic form:

$$\begin{aligned}
&\min_{x_1,\ldots,x_n} c_0 + \sum_{i=1}^{n} c_i x_i \\
&\text{s.t.} \sum_{i=1}^{n} A^e_{j,i} x_i = b^e_j, && \forall\ j = 1,\ldots,m_e \\
&\sum_{i=1}^{n} A^g_{j,i} x_i \geq b^g_j, && \forall\ j = 1,\ldots,m_g \\
&\sum_{i=1}^{n} A^l_{j,i} x_i \leq b^l_j, && \forall\ j = 1,\ldots,m_l \\
&x_i \in \mathbb{Z}, && \forall\ i = 1,\ldots,n,
\end{aligned}$$

where $m_e$, $m_g$, $m_l$, $A^e_{j,i}$, $A^g_{j,i}$, $A^l_{j,i}$, $b^e_j$, $b^g_j$, $b^l_j$, $c_0,\ldots,c_n$, and $\mathbb{Z}$ have the same interpretations as in the generic MILPP, which is given in Section 3.2.1. Thus, the only distinction between the formulation of a general MILPP and a PILPP is that *all* of the variables in a PILPP are restricted to take on integer values. Conversely, a general MILPP can include a combination of variables with and without such a restriction.

Among the examples introduced in Section 3.1, the Photovoltaic Panel-Repair Problem, which is introduced in Section 3.1.1, is a PILPP.

### 3.2.3 Mixed-Binary Linear Optimization Problems

A **mixed-binary linear optimization problem** is a special case of a general MILPP in which the variables that are restricted to take on integer values are actually further restricted to take on binary values. With rare exceptions, these binary variables are restricted to take on the values of 0 and 1 and are often used to model logical decisions or constraints.

A mixed-binary linear optimization problem has the generic form:

$$\begin{aligned}
\min_{x_1,\ldots,x_n} \quad & c_0 + \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} A^e_{j,i} x_i = b^e_j, && \forall\, j = 1,\ldots,m_e \\
& \sum_{i=1}^{n} A^g_{j,i} x_i \geq b^g_j, && \forall\, j = 1,\ldots,m_g \\
& \sum_{i=1}^{n} A^l_{j,i} x_i \leq b^l_j, && \forall\, j = 1,\ldots,m_l \\
& x_i \in \{0,1\}, && \text{for some } i = 1,\ldots,n \\
& x_i \in \mathbb{R}, && \text{for the remaining } i = 1,\ldots,n,
\end{aligned}$$

where $m_e$, $m_g$, $m_l$, $A^e_{j,i}$, $A^g_{j,i}$, $A^l_{j,i}$, $b^e_j$, $b^g_j$, $b^l_j$, and $c_0,\ldots,c_n$ have the same interpretations as in the generic MILPP, which is given in Section 3.2.1.

Of the examples given in Section 3.1, the Natural Gas-Storage, Electricity-Scheduling, and Oil-Transmission Problems, which are introduced in Sections 3.1.2, 3.1.3 and 3.1.4, respectively, are mixed-binary linear optimization problems.

### 3.2.4 *Pure-Binary Linear Optimization Problems*

We finally have the case of a **pure-binary linear optimization problem**, which is a special case of a mixed-binary linear optimization problem in which all of the variables are restricted to being binary variables. Such a problem has the form:

$$\begin{aligned}
\min_{x_1,\ldots,x_n} \quad & c_0 + \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} A^e_{j,i} x_i = b^e_j, && \forall\, j = 1,\ldots,m_e \\
& \sum_{i=1}^{n} A^g_{j,i} x_i \geq b^g_j, && \forall\, j = 1,\ldots,m_g \\
& \sum_{i=1}^{n} A^l_{j,i} x_i \leq b^l_j, && \forall\, j = 1,\ldots,m_l \\
& x_i \in \{0,1\}, && \forall\, i = 1,\ldots,n,
\end{aligned}$$

where $m_e$, $m_g$, $m_l$, $A^e_{j,i}$, $A^g_{j,i}$, $A^l_{j,i}$, $b^e_j$, $b^g_j$, $b^l_j$, $c_0,\ldots,c_n$, and $\mathbb{Z}$ have the same interpretations as in the generic MILPP, which is given in Section 3.2.1.

Of the examples introduced in Section 3.1, the Charging-Station and Wind Farm-Maintenance Problems, which are discussed in Sections 3.1.5 and 3.1.6, respectively, are examples pure-binary linear optimization problems.

## 3.3 Linearizing Nonlinearities Using Binary Variables

This section introduces one of the most powerful uses of integer optimization techniques. This is the use of integer, and in particular binary, variables to linearize complex nonlinearities and discontinuities and to formulate logical constraints in optimization problems. This is particularly useful, because it is considerably easier to solve optimization problems in which the objective function and constraints are linear in the decision variables.

Some of these linearizations are employed in formulating the examples that are discussed in Section 3.1.

### *3.3.1 Variable Discontinuity*

On occasion a variable may have a discontinuity in the sense that we would like it to be within one of two intervals. To more concretely explain this, suppose that we have a variable, $x$, and we would like $x$ to either be between $l_1$ and $u_1$ or between $l_2$ and $u_2$. We could write this restriction on $x$ as the following logical condition:

$$l_1 \leq x \leq u_1 \text{ or } l_2 \leq x \leq u_2.$$

This is not a valid linear constraint, however, because constraints in optimization problems cannot have logical statements (*i.e.*, the 'or') in them.

We can model this type of a restriction as a valid linear constraint by introducing a new binary variable, which we denote $y$. The restriction on $x$ is then written as:

$$l_1 y + l_2 \cdot (1 - y) \leq x \leq u_1 y + u_2 \cdot (1 - y). \tag{3.20}$$

Note that if $y = 0$, then (3.20) simplifies to:

$$l_2 \leq x \leq u_2,$$

and that if $y = 1$, then it becomes:

$$l_1 \leq x \leq u_1.$$

A common situation in which this type of variable discontinuity arises is when we are modeling the production of a facility that must be switched on or off. If it is

switched off, its production level must equal 0. Otherwise, if it is switched on, its production level must be between some lower and upper limits, which we denote $x^{\min}$ and $x^{\max}$. This type of a production discontinuity is illustrated in Figure 3.4. We can model this type of a restriction using (3.20) by letting $l_2 = u_2 = 0$, $l_1 = x^{\min}$, and $u_1 = x^{\max}$. In this case, Constraint (3.20) becomes:

$$x^{\min} y \leq x \leq x^{\max} y. \tag{3.21}$$


**Fig. 3.4** Production discontinuity

This technique is used in both the Natural Gas-Storage and Electricity-Scheduling Problems, which are introduced in Sections 3.1.2 and 3.1.3, respectively. More specifically, Constraints (3.6)–(3.8) in the Natural Gas-Storage Problem impose the capacities of the storage facilities and only allow each facility to store gas if it is built. Constraints (3.9)–(3.11) in the Electricity-Scheduling Problem impose the minimum and maximum production levels of the units. These constraints also restrict each production unit to produce power only if it is switched on.

It is important to stress that these types of variable discontinuities can be applied to other settings besides the modeling of production processes and facilities.

### *3.3.2 Fixed Activity Cost*

Some systems have a cost or other term in the objective function that is incurred when an activity occurs. As an example of this, consider the modeling of a production facility that must be switched on or off, as discussed in Section 3.3.1. Suppose that if the production facility is switched on, there is a fixed cost, $c_0$, that is incurred that does not depend on how many units the facility produces. In addition to this fixed cost, the facility incurs a cost of $m$ per unit produced. Thus, the total cost of operating the facility can be written as:

$$\text{cost} = \begin{cases} c_0 + mx, & \text{if facility is switched on,} \\ 0, & \text{otherwise,} \end{cases} \tag{3.22}$$

where $x$ represents the facility's production level. Figure 3.5 illustrates such a cost function where we further impose minimum and maximum production levels on the facility, in line with the example that is given in Section 3.3.1.

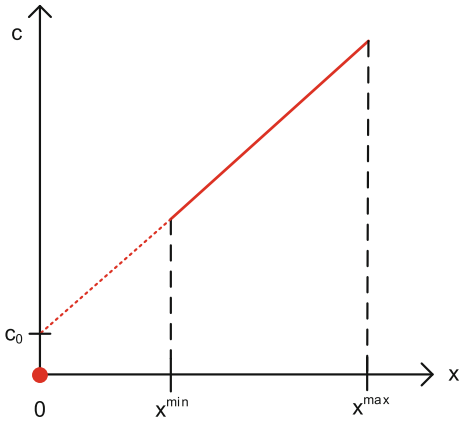


**Fig. 3.5** Fixed activity cost

Expression (3.22) is not a valid term to include in the objective function of an optimization problem, because it includes logical 'if' statements. To model this type of cost function, we introduce a binary variable, $y$, which is defined as:

$$y = \begin{cases} 1, \text{ if facility is switched on,} \\ 0, \text{ otherwise.} \end{cases}$$

The facility's cost is then modeled as:

$$\text{cost} = c_0 y + mx.$$

In nearly all cases, we include a discontinuity constraint similar to (3.21), which 'links' the decision to switch the production facility on or off (*i.e.*, the $y$ variable) to the production decision (*i.e.*, the $x$ variable). This is because if a constraint similar to (3.21) is not included, the model will allow the facility to produce units while being switched off. If there is no explicit upper bound on how many units the facility can produce, one can set $x^{\max}$ to an arbitrarily high number in (3.21). In practice, however, a real-world production facility almost invariably has an upper limit on its output.

Both the Natural Gas-Storage and Electricity-Scheduling Problems, which are introduced in Sections 3.1.2 and 3.1.3, respectively, employ this technique to model a fixed activity cost. In the Natural Gas-Storage Problem, there are binary variables representing the decision to build a storage facility. There is a fixed cost associated with building each facility that does not depend on how much gas is ultimately stored in it. Similarly, there are binary variables in the Electricity-Scheduling Problem representing the decision to switch each of the three production units on or not. Switching the production units on imposes a fixed cost that is independent of how much power is produced.

### 3.3.3 *Non-convex Piecewise-Linear Cost*

Another type of cost structure that can arise are so-called piecewise-linear costs. Figure 3.6 illustrates an example piecewise-linear cost function. The figure shows that the first $b_1$ units produced cost $\$m_1$ per unit to produce. After the first $b_1$ units, the next $(b_2 - b_1)$ units cost $\$m_2$ per unit. Finally, any remaining units after the first $b_2$ units cost $\$m_3$ per unit.

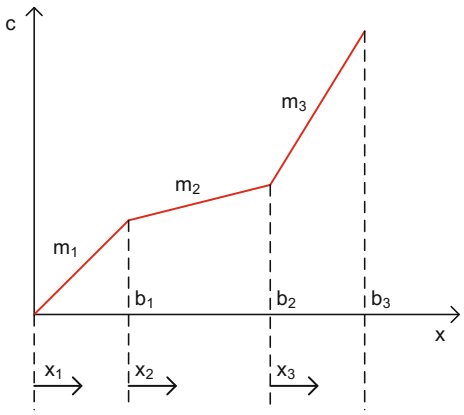


**Fig. 3.6** Non-convex piecewise-linear cost

The idea of a piecewise-linear function is that the function is linear between two breakpoints. For instance, the costs shown in Figure 3.6 are linear between 0 and $b_1$, between $b_1$ and $b_2$, and between $b_2$ and $b_3$. However, the entire function is not linear because of the 'kinks' at the breakpoints. One can write the cost function shown in Figure 3.6 as:

$$\text{cost} = \begin{cases} m_1 x, & \text{if } 0 \le x \le b_1, \\ m_1 b_1 + m_2 \cdot (x - b_1), & \text{if } b_1 < x \le b_2, \\ m_1 b_1 + m_2 b_2 + m_3 \cdot (x - b_2), & \text{otherwise}, \end{cases} \tag{3.23}$$

where we let $x$ denote the production level of the facility. This is not a valid term to include in the objective function of an optimization problem, however, because it employs logical 'if' statements.

We can linearize (3.23) by introducing three new continuous variables, $x_1, x_2$, and $x_3$. As shown in Figure 3.6, $x_1$ is defined as the number of units produced at a cost of $m_1$. The variables, $x_2$ and $x_3$, are similarly defined as the number of units produced at costs of $m_2$ and $m_3$, respectively. We next define two binary variables, $y_1$ and $y_2$, as:

$$y_i = \begin{cases} 1, & \text{if the maximum units that can be produced at a cost of } m_i \text{ are produced,} \\ 0, & \text{otherwise.} \end{cases}$$

We then model the production cost as:

$$\text{cost} = m_1x_1 + m_2x_2 + m_3x_3,$$

and include the constraints:

$$x = x_1 + x_2 + x_3 \tag{3.24}$$

$$b_1y_1 \leq x_1 \leq b_1 \tag{3.25}$$

$$(b_2 - b_1)y_2 \leq x_2 \leq (b_2 - b_1)y_1 \tag{3.26}$$

$$0 \leq x_3 \leq (b_3 - b_2)y_2 \tag{3.27}$$

$$y_1 \leq y_2$$

$$y_1, y_2 \in \{0, 1\},$$

in the optimization model.

To understand the logic behind this set of constraints, first note that because of the constraint $y_1 \leq y_2$ we only have three possible cases for the values of the $y$ variables. In the first, in which $y_1 = y_2 = 0$, (3.24)–(3.27) reduce to:

$$x = x_1 + x_2 + x_3$$

$$0 \leq x_1 \leq b_1$$

$$0 \leq x_2 \leq 0$$

$$0 \leq x_3 \leq 0,$$

meaning that we have $x_2 = x_3 = 0$ and $x = x_1$ can take on any value between 0 and $b_1$.

Next, if $y_1 = 1$ and $y_2 = 0$, then (3.24)–(3.27) become:

$$x = x_1 + x_2 + x_3$$

$$b_1 \leq x_1 \leq b_1$$

$$0 \leq x_2 \leq b_2 - b_1$$

$$0 \leq x_3 \leq 0,$$

meaning that we have $x_1 = b_1$, $x_3 = 0$, and $x_2$ can take on any value between 0 and $(b_2 - b_1)$.

Finally, in the case in which $y_1 = y_2 = 1$, Constraints (3.24)–(3.27) become:

$$x = x_1 + x_2 + x_3$$

$$b_1 \leq x_1 \leq b_1$$

$$b_2 - b_1 \leq x_2 \leq b_2 - b_1$$

$$0 \leq x_3 \leq b_3 - b_2,$$

meaning that we have $x_1 = b_1$, $x_2 = b_2$, and that $x_3$ can take on any value between 0 and $(b_3 - b_2)$.

Note that we can further generalize this technique to model a piecewise-linear function with any arbitrary number of pieces (unlike the three pieces assumed in Figure 3.6). To see this, suppose that $x$ measures the total production level being modeled and that per-unit production costs $m_1, m_2, \ldots, m_N$ apply to different production levels with breakpoints at $b_1, b_2, \ldots, b_N$. We would model the production cost by introducing $N$ variables, denoted $x_1, x_2, \ldots, x_N$, and $(N - 1)$ binary variables, denoted $y_1, y_2, \ldots, y_{N-1}$. We would model the production cost as:

$$\text{cost} = \sum_{i=1}^{N} m_i x_i,$$

and add the constraints:

$$x = \sum_{i=1}^{N} x_i$$

$$b_1 y_1 \leq x_1 \leq b_1$$

$$(b_2 - b_1) y_2 \leq x_2 \leq (b_2 - b_1) y_1$$

$$(b_3 - b_2) y_3 \leq x_3 \leq (b_3 - b_2) y_2$$

$$\vdots$$

$$(b_{N-1} - b_{N-1}) y_{N-1} \leq x_{N-1} \leq (b_{N-1} - b_{N-1}) y_{N-2}$$

$$0 \leq x_N \leq (b_N - b_{N-1}) y_{N-1}$$

$$y_1 \leq y_2 \leq y_3 \leq \cdots \leq y_{N-1}$$

$$y_i \in \{0, 1\}, \forall i = 1, \ldots, N_1,$$

to the model.

We finally note that this use of binary variables is generally needed if one is modeling a piecewise-linear function that is *not* convex (*cf.* Appendix B for further discussion of convex functions). The piecewise-linear function depicted in Figure 3.6 is non-convex because the per-unit costs, $m_1$, $m_2$, and $m_3$, are not non-decreasing (*i.e.*, we do not have $m_1 \leq m_2 \leq m_3$). Otherwise, if the per-unit costs are non-decreasing, the piecewise-linear cost function is convex. In that case, if the convex piecewise-linear cost function is being minimized, binary variables are not needed.

To see this, return to the general case in which $x$ measures the total production level being modeled and that per-unit production costs $m_1, m_2, \ldots, m_N$ apply to different production levels with breakpoints at $b_1, b_2, \ldots, b_N$. Moreover, assume that we have $m_1 \leq m_2 \leq \cdots \leq m_N$. We would model the production cost by introducing $N$ variables, denoted $x_1, x_2, \ldots, x_N$. The production cost is then computed as:

$$\text{cost} = \sum_{i=1}^{N} m_i x_i,$$

and we would add the constraints:

$$x = \sum_{i=1}^{N} x_i$$

$$0 \leq x_1 \leq b_1$$

$$0 \leq x_2 \leq b_2 - b_1$$

$$\vdots$$

$$0 \leq x_N \leq b_N - b_{N-1},$$

to the model.

We do not need to use binary variables in this case because it is optimal to fully exhaust production in level $i$ before using any production in level $(i+1)$. The reason for this is that production in level $(i+1)$ (and all subsequent levels) is more costly than production in level $i$. The example that is given in Figure 3.6, with the non-convex cost, does not exhibit this property. Without the binary variables included, the model would choose to use production level 2 (at a per-unit cost of $m_2$) before production level 2 (with a higher per-unit cost of $m_1$) is used.

### *3.3.4 Alternative Constraints*

In some situations, we may be interested in enforcing either one constraint:

$$\sum_{i=1}^{n} a_{1,i} x_i \leq b_1, \tag{3.28}$$

or an alternative one:

$$\sum_{i=1}^{n} a_{2,i} x_i \leq b_2, \tag{3.29}$$

but not both. As discussed in Section 3.3.1, the constraint:

$$\sum_{i=1}^{n} a_{1,i} x_i \leq b_1 \text{ or } \sum_{i=1}^{n} a_{2,i} x_i \leq b_2, \tag{3.30}$$

is not valid in an optimization problem, because it includes a logical 'or' statement. We can, however, linearize the 'or' statement through the use of a binary variable.

To do this, we define the binary variable, $y$, as:

$$y = \begin{cases} 1, \text{ if the constraint } \sum_{i=1}^{n} a_{1,i} x_i \leq b_1 \text{ is enforced,} \\ 0, \text{ otherwise.} \end{cases}$$

We then replace (3.30) with:

$$\sum_{i=1}^{n} a_{1,i} x_i \leq b_1 + M_1 \cdot (1 - y) \tag{3.31}$$

$$\sum_{i=1}^{n} a_{2,i} x_i \leq b_2 + M_2 y, \tag{3.32}$$

where $M_1$ and $M_2$ are sufficiently large constants.

To see how this formulation works, let us suppose that we have $y = 1$. If so, then (3.31) becomes:

$$\sum_{i=1}^{n} a_{1,i} x_i \leq b_1,$$

meaning that Constraint (3.28) is being enforced on the optimization problem. On the other hand, Constraint (3.32) becomes:

$$\sum_{i=1}^{n} a_{2,i} x_i \leq b_2 + M_2.$$

Note that if $M_2$ is sufficiently large, then any values for $x$ will satisfy this constraint, meaning that Constraint (3.29) is not being imposed on the problem. In the case in which $y = 0$, Constraint (3.31) becomes:

$$\sum_{i=1}^{n} a_{1,i} x_i \leq b_1 + M_1,$$

meaning that Constraint (3.28) is not being imposed, so long as $M_1$ is sufficiently large. On the other hand, (3.32) becomes:

$$\sum_{i=1}^{n} a_{2,i} x_i \leq b_2,$$

meaning that Constraint (3.29) is being enforced.

### *3.3.5 Product of Two Variables*

A very powerful use of binary variables is to linearize the product of two variables in an optimization problem. We discuss here how to linearize the product of variables in three different cases. We first discuss linearizing the product of a real and binary variable and then the product of two binary variables, both of which can be linearized exactly. We then discuss the use of a technique, known as binary expansion, to linearly approximate the product of two real variables.

#### 3.3.5.1 Product of a Real and Binary Variable

Here we demonstrate how to linearize the product of a real and binary variable. Suppose that $x \in \mathbb{R}$ is a real variable while $y \in \{0, 1\}$ is a binary variable. If we define $p = xy$ as the product of these two variables, then we know that $p$ can take on only the following two values:

$$p = \begin{cases} 0, & \text{if } y = 0, \\ x, & \text{otherwise.} \end{cases} \tag{3.33}$$

To come up with a linear expression for $p$, we must further assume that $x$ is bounded:

$$-l \leq x \leq u.$$

If there are no explicit bounds on $x$, one can impose bounds on $x$ in an optimization problem by making $l$ and $u$ sufficiently small and large, respectively. We must also add a new real variable, which we denote as $z$. We can then define $p$ through the following constraints:

$$p = x - z \tag{3.34}$$

$$-ly \leq p \leq uy \tag{3.35}$$

$$-l \cdot (1-y) \leq z \leq u \cdot (1-y). \tag{3.36}$$

To see how this linearization works, first consider the case in which $y = 0$. If so, Constraints (3.34)–(3.36) simplify to:

$$p = x - z$$

$$0 \leq p \leq 0$$

$$-l \leq z \leq u,$$

meaning that $p = 0$, which is consistent with (3.33), and that $z = x$. The variable $z$ is essentially playing the role of a slack variable in this formulation. Otherwise, if $y = 1$, then (3.34)–(3.36) become:

$$p = x - z$$

$$-l \leq p \leq u$$

$$0 \leq z \leq 0,$$

in which case $z = 0$ and $p = x$, as required by (3.33).

#### 3.3.5.2 Product of Two Binary Variables

We now consider the case in which two binary variables are being multiplied. Let us suppose that $x, y \in \{0, 1\}$ are two binary variables and define $p = xy$ as their product. We know that $p$ can take on one of two values:

$$p = \begin{cases} 1, \text{ if } x = 1 \text{ and } y = 1, \\ 0, \text{ otherwise.} \end{cases}$$

We can express $p$ linearly through the following set of constraints:

$$p \leq x \tag{3.37}$$

$$p \leq y \tag{3.38}$$

$$p \geq 0 \tag{3.39}$$

$$p \geq x + y - 1. \tag{3.40}$$

To see how this linearization works, first consider the case in which both $x = 1$ and $y = 1$. If so, then (3.37)–(3.40) become:

$$p \leq 1$$

$$p \leq 1$$

$$p \geq 0$$

$$p \geq 1,$$

which forces $p = 1$. Otherwise, if at least one of $x$ or $y$ is equal to zero, then (3.37)–(3.40) force $p = 0$.

#### 3.3.5.3 Product of Two Real Variables

The product of two real variables cannot be linearized exactly. However, there is a technique, known as **binary expansion**, that can be used to linearly *approximate* the product of two real variables. To demonstrate the concept of binary expansion, let us define $x, y \in \mathbb{R}$ as two real variables, and $p = xy$ as their product.

To linearize the product, we must approximate one of the two variables as taking on one of a finite number of values. Let us suppose that we approximate $y$ as taking on one of the $N$ values $y_1, y_2, \ldots, y_N$, where $y_1, y_2, \ldots, y_N$ are fixed constants. To conduct the binary expansion, we introduce $N$ binary variables, which we denote $z_1, z_2, \ldots, z_N$. We then approximate $y$ using the following:

$$y \approx \sum_{i=1}^{N} y_i z_i$$

$$\sum_{i=1}^{N} z_i = 1.$$

Note that because exactly one of the $z_i$'s must equal 1, $y$ is approximated as taking on the corresponding value of $y_i$.

Using this approximation of $y$, we can then approximate the product of $x$ and $y$ by adding the following constraints to the optimization problem:

$$p = \sum_{i=1}^{N} y_i z_i x \tag{3.41}$$

$$\sum_{i=1}^{N} z_i = 1$$

$$z_i \in \{0, 1\}, \forall i = 1, \ldots, N.$$

Because the $y_i$'s are constants, the right-hand side of (3.41) involves the product of a binary and real variable. This must then be linearized using the technique that is outlined in Section 3.3.5.1.

It is important to stress once again that binary expansion does not exactly represent the product of two real variables. Rather, it approximates the product. Nevertheless, it can be a very useful technique. How good of an approximation binary expansion provides depends on whether the 'true' value of $y$ is close to one of the $y_i$'s. If so, the approximation will be better. Because we do not typically know *a priori* what the 'true' value of $y$ is, we deal with this issue by using a large number of $y_i$'s. Doing so increases the size of the problem, however, because more binary variables (*i.e.*, $z_i$'s) must be introduced into the model. Increasing the size of the problem invariably makes it more difficult to solve.

## 3.4 Relaxations

This section introduces a very important concept in solving mixed-integer linear optimization problems. This is the idea of a **relaxation**. A relaxation of an optimization problem is a problem in which one or more of the constraints are loosened, relaxed, or entirely removed. When the constraint is relaxed, the feasible region of the problem grows in size. As a result of this, there are some important properties linking a problem and a relaxation that are useful when we solve MILPPs.

All of this discussion assumes that we have a generic MILPP, which can be written as:

$$\min_{x_1,\ldots,x_n} c_0 + \sum_{i=1}^{n} c_i x_i \tag{3.42}$$

$$\begin{aligned}
\text{s.t.} \quad & \sum_{i=1}^{n} A^e_{j,i} x_i = b^e_j, && \forall\ j = 1, \ldots, m_e \\
& \sum_{i=1}^{n} A^g_{j,i} x_i \geq b^g_j, && \forall\ j = 1, \ldots, m_g \\
& \sum_{i=1}^{n} A^l_{j,i} x_i \leq b^l_j, && \forall\ j = 1, \ldots, m_l \\
& x_i \in \mathbb{Z}, && \text{for some } i = 1, \ldots, n \\
& x_i \in \mathbb{R}, && \text{for the remaining } i = 1, \ldots, n.
\end{aligned} \tag{3.43}$$

This generic form captures all of the special cases of MILPPs that are introduced in Section 3.2. Moreover, we know from the discussion in Section 2.2.2.1 that a MILPP

that is a maximization can be converted to a minimization simply by multiplying the objective function by $-1$.

We now introduce what is known as the linear relaxation of this MILPP, which is:

$$\begin{aligned}
&\min_{x_1,\ldots,x_n} c_0 + \sum_{i=1}^{n} c_i x_i \\
&\text{s.t.} \sum_{i=1}^{n} A^e_{j,i} x_i = b^e_j, && \forall\ j = 1,\ldots,m_e \\
&\sum_{i=1}^{n} A^g_{j,i} x_i \geq b^g_j, && \forall\ j = 1,\ldots,m_g \\
&\sum_{i=1}^{n} A^l_{j,i} x_i \leq b^l_j, && \forall\ j = 1,\ldots,m_l \\
&x_i \in \mathbb{R}, && \forall\ i = 1,\ldots,n.
\end{aligned}$$

The only difference between the original MILPP and its linear relaxation is that Constraint (3.43) is relaxed, because we allow *all* of the variables to be real-valued in the relaxed problem. This can be contrasted with the original MILPP, in which some of the variables are restricted to take on only integer values.

It is important to stress that there are many ways in which an optimization problem can be relaxed. When writing the linear relaxation, we remove the constraints that the variables be integer-valued. However, one could, for example, relax the MILPP by removing the equality and greater-than-or-equal-to constraints, which would give the following relaxed problem:

$$\begin{aligned}
&\min_{x_1,\ldots,x_n} c_0 + \sum_{i=1}^{n} c_i x_i \\
&\text{s.t.} \sum_{i=1}^{n} A^l_{j,i} x_i \leq b^l_j, && \forall\ j = 1,\ldots,m_l \\
&x_i \in \mathbb{Z}, && \text{for some } i = 1,\ldots,n \\
&x_i \in \mathbb{R}, && \text{for the remaining } i = 1,\ldots,n.
\end{aligned}$$

Moreover, one can relax constraints in any type of optimization problem (including linear optimization problems).

We can now show three useful relationships between the original problem and its relaxation. Note that these relationships apply to any problem and a relaxation. However, to make the results more clear, we will focus on the generic MILPP and its linear relaxation.

**Relaxation-Optimality Property:** The optimal objective-function value of the linear relaxation is less than or equal to the optimal objective-function value of the original MILPP.

Suppose that $x^*$ is an optimal solution to the original MILPP. By definition, $x^*$ is feasible in the MILPP. $x^*$ is also feasible in the linear relaxation. This is because the linear relaxation has the same equality and inequality constraints as the original MILPP, but does not have the integrality constraints. There may, however, be a solution, $\tilde{x}$, that is feasible in the linear relaxation that gives a smaller objective-function value than $x^*$. $\tilde{x}$ is not feasible in the original MILPP (if it is, then $x^*$ would not be an optimal solution to the MILPP). Thus, the linear relaxation may have an optimal solution that is not feasible in the original MILPP and gives a smaller objective-function value than $x^*$.

**Relaxation-Optimality Corollary:** If the optimal solution of the linear relaxation satisfies the constraints of the original MILPP, then this solution is also optimal in the original MILPP.

This result comes immediately from the Relaxation-Optimality Property. Suppose that $x^*$ is optimal in the linear relaxation and that it is feasible in the original MILPP. We know from the Relaxation-Optimality Property that the optimal objective-function value of the original MILPP can be no lower than the objective-function value given by $x^*$. Combining this observation with the fact that $x^*$ is feasible in the original MILPP tells us that $x^*$ is also optimal in the original MILPP.

**Relaxation-Feasibility Property:** If the linear relaxation of the original MILPP is infeasible, then the original MILPP is also infeasible.

We show this by contradiction. To do so, we suppose that the property is not true. This would mean that the linear relaxation is infeasible but the original MILPP is feasible. If so, then there is a solution, $\hat{x}$, that is feasible in the original MILPP. $\hat{x}$ must also be feasible in the linear relaxation, however. This is because the linear relaxation has the same equality and inequality constraints as the original MILPP but does not have the integrality constraints. Thus, it is impossible for the Relaxation-Feasibility Property to not hold.

We use these three properties in Section 3.5 to outline an effective algorithm for solving general MILPPs.

## 3.5 Solving Mixed-Integer Linear Optimization Problems Using Branch and Bound

We describe in this section the **Branch-and-Bound Algorithm**, which can be used to efficiently solve MILPPs. We begin by first providing a high-level motivation behind the algorithm. We next outline the steps of the algorithm and then illustrate its use with a simple example. We finally provide a more formal outline of the algorithm.

All of this discussion assumes that we have a generic MILPP that is in the form:

$$
\begin{aligned}
\min_{x_1,\ldots,x_n} \quad & c_0 + \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} A^e_{j,i} x_i = b^e_j, && \forall\ j = 1, \ldots, m_e \\
& \sum_{i=1}^{n} A^g_{j,i} x_i \geq b^g_j, && \forall\ j = 1, \ldots, m_g \\
& \sum_{i=1}^{n} A^l_{j,i} x_i \leq b^l_j, && \forall\ j = 1, \ldots, m_l \\
& x_i \in \mathbb{Z}, && \text{for some } i = 1, \ldots, n \\
& x_i \in \mathbb{R}, && \text{for the remaining } i = 1, \ldots, n.
\end{aligned}
$$

### *3.5.1 Motivation*

At a high level, the Branch-and-Bound Algorithm works through the following four steps.

1. The algorithm starts by solving the linear relaxation of the original MILPP.
2. Based on the solution of the linear relaxation, the algorithm generates a sequence of additional optimization problems in which constraints are added to the linear relaxation.
3. As this sequence of additional optimization problems is solved, the algorithm establishes upper and lower bounds on the optimal objective-function value of the original MILPP. The upper bound progressively decreases while the lower bound increases.
4. *Ideally*, this process continues until the full sequence of additional optimization problems are solved, which gives an optimal solution. In practice, we may stop once we have found a solution that is feasible in the original MILPP and that has upper and lower bounds that are close enough to one another (indicating that the feasible solution is very close to optimal).

The lower bound on the optimal objective-function value is found by appealing to the Relaxation-Optimality Property, which is discussed in Section 3.4. We know from the Relaxation-Optimality Property that the solution to the linear relaxation provides a lower bound on the optimal objective-function value of the original MILPP.

We obtain upper bounds whenever solving one of the additional optimization problems gives a solution that is feasible in the MILPP. We can reason that such a solution gives an upper bound on the optimal objective-function value. This is because of the definition of an optimal solution as being a feasible solution that gives the best objective-function value. If we find a solution that is feasible, it may be optimal. However, it may not be, in which case the objective-function value of the feasible solution is greater than the optimal objective-function value.

### *3.5.2 Outline of Branch-and-Bound Algorithm*

Building off of the high-level motivation given in Section 3.5.1, we now outline the major steps of the Branch-and-Bound Algorithm in further detail.

#### 3.5.2.1 Step 0: Initialization

We begin the Branch-and-Bound Algorithm by initializing it. Throughout the algorithm, we let $z^l$ denote the current lower bound and $z^u$ the current upper bound on the optimal objective-function value. We also keep track of a set, which we denote as $\Xi$, of linear optimization problems that remain to be solved. As we solve the sequence of linear optimization problems in $\Xi$, we also keep track of the best solution (in the sense of giving the smallest objective-function value) that has been found so far that satisfies all of the integrality constraints of the original MILPP. We denote this by $x^b$.

We initialize the bounds by letting $z^l \leftarrow -\infty$ and $z^u \leftarrow +\infty$. The reasoning behind this is that we have, as of yet, not done any work to solve the original MILPP. Thus, all we know is that its optimal objective-function value is between $-\infty$ and

$+\infty$. We also set $\Xi \leftarrow \emptyset$, because we have not generated any linear optimization problems to solve yet. We do not initialize $x^b$ because we have not yet found a solution that satisfies the integrality constraints of the original MILPP.

We next solve the linear relaxation of the MILPP. When we solve the linear relaxation, one of following four outcomes are possible.

1. The linear relaxation may be infeasible. If so, then based on the Relaxation-Feasibility Property, the original MILPP is infeasible as well. As such, we terminate the Branch-and-Bound Algorithm and report that the original MILPP is infeasible.
2. Solving the linear relaxation may give a solution that satisfies all of the integrality constraints of the original MILPP. If so, then based on the Relaxation-Optimality Corollary, the optimal solution of the linear relaxation is also optimal in the original MILPP. Thus, we terminate the Branch-and-Bound Algorithm and report the solution found as being optimal.
3. Solving the linear relaxation may give a solution that does not satisfy all of the integrality constraints of the original MILPP. If so, we know from the Relaxation-Optimality Property that the optimal objective-function value of the linear relaxation is a lower bound on the optimal objective-function value of the original MILPP. Thus, we update the current lower bound as $z^l \leftarrow z^0$, where $z^0$ denotes the optimal objective-function value of the linear relaxation. We then continue with Step 1 of the algorithm.
4. The linear relaxation may be unbounded. If so, we cannot necessarily conclude whether the original MILPP is unbounded or not. We can only conclude that the original MILPP is unbounded if we can find a solution that is feasible in the original MILPP and that makes the objective function go to $-\infty$. If so, we terminate the algorithm and report that the original MILPP is unbounded. Otherwise, we continue with Step 1 of the algorithm.

#### 3.5.2.2 Step 1: Initial Branching

Let $x^0$ denote the optimal solution to the linear relaxation, found in Step 0. Pick one of the variables that is supposed to be integer-valued in the original MILPP but has a non-integer value in $x^0$. We hereafter call this chosen variable $x_i$.

We generate two new linear optimization problems in which we add a new constraint to the linear relaxation. The first one, which we denote $\text{LPP}_1$, consists of the linear relaxation with the added constraint:

$$x_i \leq \lfloor x_i^0 \rfloor.$$

The notation $\lfloor x_i^0 \rfloor$ denotes the floor of $x_i^0$, which means that we round $x_i^0$ down to the nearest integer less than $x_i^0$, which is the value for $x_i$ in the optimal solution to the linear relaxation. The second linear optimization problem, which we denote $\text{LPP}_2$, consists of the linear relaxation with the added constraint:

$$x_i \geq \lceil x_i^0 \rceil.$$

The notation $\lceil x_i^0 \rceil$ denotes the ceiling of $x_i^0$, meaning that we round $x_i^0$ up to the nearest integer greater than $x_i^0$.

After generating these two new linear optimization problems, we update the set of linear optimization problems that remain to be solved as:

$$\Xi \leftarrow \Xi \cup \text{LPP}_1 \cup \text{LPP}_2.$$

These two new problems, $\text{LPP}_1$ and $\text{LPP}_2$, cover the entire feasible region of the original MILPP. The reason for this is that all we have done in generating $\text{LPP}_1$ and $\text{LPP}_2$ is created two new problems, with $x_i$ being constrained to take on values less than $\lfloor x_i^0 \rfloor$ in one and values greater than $\lceil x_i^0 \rceil$ in the other. Thus, the only points that are no longer feasible in these two LPPs are those in which $x_i$ takes on a value strictly between $\lfloor x_i^0 \rfloor$ and $\lceil x_i^0 \rceil$. However, such points are infeasible in the original MILPP, because there are no integer values between $\lfloor x_i^0 \rfloor$ and $\lceil x_i^0 \rceil$ and $x_i$ is restricted to taking on integer values in the original MILPP.

#### 3.5.2.3 Step 2: Solving

Select a linear optimization problem in $\Xi$, solve it, and remove it from $\Xi$. Let $\hat{x}$ denote the optimal solution to this problem and let $\hat{z}$ denote its optimal objective-function value.

#### 3.5.2.4 Step 3: Bound Updating and Branching

When we solve the linear optimization problem from $\Xi$ chosen in Step 2, one of the four following outcomes are possible. In this step, we can update the bounds and may need to add more linear optimization problems to $\Xi$, based on the different possible outcomes.

1. $\hat{x}$ may satisfy all of the integrality constraints of the original MILPP. This means that $\hat{x}$ is feasible in the original MILPP. We further know, from the discussion in Section 3.5.1, that $\hat{z}$ provides an upper bound on the optimal objective-function value of the original MILPP.

   If $\hat{z} < z^u$, then this means that $\hat{x}$ is the best solution that is feasible in the original MILPP that we have found thus far. In this case, we update the upper bound $z^u \leftarrow \hat{z}$ and the best feasible solution found thus far $x^b \leftarrow \hat{x}$.

   Otherwise, if $\hat{z} \geq z^u$, the upper bound cannot be updated and we proceed to Step 4.
2. The problem solved in Step 2 may be infeasible. In this case, the bounds cannot be updated and we proceed to Step 4.

3. $\hat{x}$ may not satisfy all of the integrality constraints of the original MILPP. If so, $\hat{z}$ may provide an updated lower bound on the optimal objective-function value of the original MILPP. We may also need to generate two new linear optimization problems to add to $\Xi$.

   The lower bound on the optimal objective-function value of the original MILPP can be updated if $z^l < \hat{z} \leq z^u$ and $z^u < +\infty$. If both of these conditions are met, then we can update the lower bound as $z^l \leftarrow \hat{z}$. Otherwise, we cannot update the lower bound. We cannot update the bound in this case because the upper bound is still $+\infty$, which is not a valid bounding reference.

   Moreover, if $\hat{z} < z^u$, then we must also generate two new linear optimization problems. This is done in a manner similar to the Initial Branching in Step 1. To do this, we select one of the variables that is supposed to be integer-valued in the original MILPP but has a non-integer value in $\hat{x}$. We hereafter call this chosen variable, $x_i$. We generate two new linear optimization problems, in which we add a new constraint to the linear optimization problem most recently solved in Step 2 (*i.e.*, the problem that has $\hat{x}$ as an optimal solution). The first problem has the constraint:

$$x_i \leq \left\lfloor \hat{x}_i \right\rfloor,$$

   while the second has the constraint:

$$x_i \geq \left\lceil \hat{x}_i \right\rceil,$$

   added. These two problems are added to $\Xi$. We then proceed to Step 4. Note that for the same reason as with the Initial Branching step, these two new problems that are added to $\Xi$ cover the entire feasible region of the linear optimization problem most recently solved in Step 2.

   If $\hat{z} \geq z^u$, then we do not need to add any problems to $\Xi$ and we proceed to Step 4. The reason we do not add any problems to $\Xi$ is that no better solution than the current best one (*i.e.*, $x^b$) can be found from the problems generated from the linear optimization problem most recently solved in Step 2.
4. The problem solved in Step 2 may be unbounded. If so, we cannot necessarily conclude that the original MILPP is unbounded. We can only conclude that the original MILPP is unbounded if we can find a solution that is feasible in the original MILPP and that makes the objective function go to $-\infty$. If so, we terminate the algorithm and report that the original MILPP is unbounded. Otherwise, we generate two new linear optimization problems, following the process outlined in Case 3 and then proceed to Step 4.

#### 3.5.2.5 Step 4: Optimality Check

We finally, in this step, determine if the Branch-and-Bound Algorithm has more problems to solve or if the algorithm can terminate. If $\Xi \neq \emptyset$, that means that there

are more linear optimization problems to be solved. In this case, we return to Step 2, select a new problem from $\Xi$ to solve, and continue with the algorithm.

If $\Xi = \emptyset$ and a value has been assigned to $x^b$, this means that we have found a feasible solution in the original MILPP and that there are no remaining feasible solutions in the original MILPP that give a better objective-function value than $x^b$. In this case, we terminate the algorithm and report that $x^b$ is an optimal solution to the original MILPP.

The final case is that $\Xi = \emptyset$ and no value has been assigned to $x^b$. This means that we are not able to find a feasible solution to the MILPP and that the MILPP is infeasible. Thus, we terminate the algorithm and report this.

We now demonstrate the use of the Branch-and-Bound Algorithm in the following example.

*Example 3.1* Consider the Photovoltaic Panel-Repair Problem, which is introduced in Section 3.1.1. This problem can be formulated as:

$$\begin{aligned} \min_{x_1,x_2} z &= -x_1 - x_2 \\ \text{s.t. } & 17x_1 + 32x_2 \leq 136 \\ & 32x_1 + 15x_2 \leq 120 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}, \end{aligned}$$

where the objective function has been changed to a minimization by multiplying through by $-1$.

To solve this problem using the Branch-and-Bound Algorithm, we first initialize:

$$z^l \leftarrow -\infty,$$

$$z^u \leftarrow +\infty,$$

and:

$$\Xi = \emptyset.$$

We next solve the linear relaxation of the original MILPP, which is:

$$\begin{aligned} \min_{x_1,x_2} z &= -x_1 - x_2 \\ \text{s.t. } & 17x_1 + 32x_2 \leq 136 \\ & 32x_1 + 15x_2 \leq 120 \\ & x_1, x_2 \geq 0. \end{aligned}$$

The optimal solution to this problem is $x^0 = (2.341, 3.007)^\top$ with $z^0 = -5.347$. This solution to the linear relaxation falls under Case 3 of the initialization step of the algorithm (we have a bounded feasible solution to the linear relaxation that *does not* satisfy the integrality constraints of the original MILPP). Thus, we update the lower bound:

$$z^l \leftarrow z^0 = -5.347.$$

This initialization step and the relaxation of the original MILPP into its linear relaxation are illustrated in Figure 3.7, where we denote the linear relaxation as '$\text{LPP}_0$'.

$$\text{MILPP} \longrightarrow \text{LPP}_0$$

**Fig. 3.7** Initialization of the Branch-and-Bound Algorithm

We next go to the Initial Branching step. To do this, we pick one of the variables that has a non-integer value in $x^0$ but must be integer-valued in the original MILPP. In this case, we can choose either of $x_1$ or $x_2$ to branch on, and arbitrarily pick $x_1$ for this example. We form two new linear optimization problems from the linear relaxation. The first one will have the constraint:

$$x_1 \leq \lfloor x_1^0 \rfloor = 2,$$

added. Thus, this problem, which we call $\text{LPP}_1$, is:

$$\begin{aligned} \min_{x_1, x_2} \; & z = -x_1 - x_2 \\ \text{s.t. } & 17x_1 + 32x_2 \leq 136 \\ & 32x_1 + 15x_2 \leq 120 \\ & x_1, x_2 \geq 0 \\ & \boldsymbol{x_1 \leq 2}. \end{aligned}$$

The second problem will have the constraint:

$$x_1 \geq \lceil x_1^0 \rceil = 3,$$

added. Thus, this problem, which we call $\text{LPP}_2$, is:

$$\begin{aligned} \min_{x_1, x_2} \; & z = -x_1 - x_2 \\ \text{s.t. } & 17x_1 + 32x_2 \leq 136 \\ & 32x_1 + 15x_2 \leq 120 \\ & x_1, x_2 \geq 0 \\ & \boldsymbol{x_1 \geq 3}. \end{aligned}$$

We then update the set of problems to be solved as:

$$\Xi \leftarrow \Xi \cup \text{LPP}_1 \cup \text{LPP}_2 = \{\text{LPP}_1, \text{LPP}_2\}.$$

Figure 3.8 shows the two new linear optimization problems being added to the set $\Xi$. It also illustrates why we refer to the process of adding new optimization problems as 'branching.' What we are essentially doing in creating these new problems is adding some restrictions (the new constraint added) to the most recently solved problem. This results in the problems forming something of a tree, as more and more constraints are added to problems that are solved (this tree-like structure is further shown in Figures 3.9–3.16). $\text{LPP}_0$ is shaded in Figure 3.8 to illustrate that this problem is processed and no longer needs to be solved at this point. We use this convention throughout Figures 3.9–3.16 to indicate problems that have been processed.

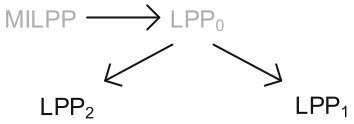


**Fig. 3.8** Initial Branching in the Branch-and-Bound Algorithm

We next proceed to the Solving step and select one of the problems from $\Xi$ to solve. In theory, we can select any of the problems in $\Xi$. As discussed in Section 3.5.5, there are different 'strategies' that can be employed to determine which problem in $\Xi$ to solve. We choose to solve $\text{LPP}_2$ at this point, which has the optimal solution $\hat{x} = (3, 1.67)^\top$ and objective-function value $\hat{z} = -4.6$. We also update the set of problems remaining to be solved by removing $\text{LPP}_2$, which gives:

$$\Xi \leftarrow \{\text{LPP}_1, \cancel{\text{LPP}_2}\} = \{\text{LPP}_1\}.$$

We now proceed to the Bound Updating and Branching step. The solution to $\text{LPP}_2$ falls into Case 3 of Step 3, thus we first check to see if we can update the lower bound, $z^l$. We find that $z^l \leq \hat{z} \leq z^u$, however because $z^u = +\infty$, we cannot update the lower bound. Next, because we have that $\hat{z} \leq z^u$, we know that we must add two new optimization problems to $\Xi$. To do this, we pick one of the variables that must be integer-valued in the original MILPP, but which has a non-integer value in $\hat{x}$. There is only one option for this, which is $x_2$. Thus, we create two new problems in which we add the constraints:

$$x_2 \leq \lfloor \hat{x}_2 \rfloor = 1,$$

and:

$$x_2 \geq \lceil \hat{x}_2 \rceil = 2,$$

to $\text{LPP}_2$. These two new problems, which we denote $\text{LPP}_3$ and $\text{LPP}_4$, are:

$$\begin{aligned}
\min_{x_1,x_2} z &= -x_1 - x_2 \\
\text{s.t. } & 17x_1 + 32x_2 \leq 136 \\
& 32x_1 + 15x_2 \leq 120 \\
& x_1, x_2 \geq 0 \\
& x_1 \geq 3 \\
& \boldsymbol{x_2 \leq 1},
\end{aligned}$$

and:

$$\begin{aligned}
\min_{x_1,x_2} z &= -x_1 - x_2 \\
\text{s.t. } & 17x_1 + 32x_2 \leq 136 \\
& 32x_1 + 15x_2 \leq 120 \\
& x_1, x_2 \geq 0 \\
& x_1 \geq 3 \\
& \boldsymbol{x_2 \geq 2},
\end{aligned}$$

respectively. The set of problems remaining to be solved is then updated:

$$\Xi \leftarrow \Xi \cup \text{LPP}_3 \cup \text{LPP}_4 = \{\text{LPP}_1, \text{LPP}_3, \text{LPP}_4\}.$$

Figure 3.9 shows the resulting tree structure of the problems when the two new LPPs are added to $\Xi$.

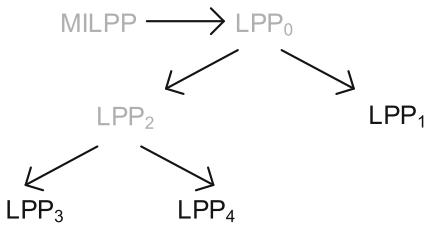


**Fig. 3.9** First Bound Updating and Branching in the Branch-and-Bound Algorithm

We next proceed to the Optimality Checking step. Because $\Xi \neq \emptyset$, the Branch-and-Bound Algorithm does *not* terminate and we instead return to Step 2.

In Step 2 we select a problem in $\Xi$ to solve. We arbitrarily select $\text{LPP}_3$, which has optimal solution $\hat{x} = (3.281, 1)^\top$ and objective-function value $\hat{z} = -4.281$. We then remove $\text{LPP}_3$ from $\Xi$, giving:

$$\Xi \leftarrow \{\text{LPP}_1, \cancel{\text{LPP}_3}, \text{LPP}_4\} = \{\text{LPP}_1, \text{LPP}_4\},$$

and proceed to the Bound Updating and Branching step.

The solution to $\text{LPP}_3$ again falls into Case 3. We next find that because $z^l \leq \hat{z} \leq z^u$ but $z^u = +\infty$ we still cannot update the lower bound. Moreover, because $\hat{z} \leq z^u$ we must add two new optimization problems to $\Xi$. We choose $x_1$ as the variable on which to add constraints in these new problems. More specifically, the two new problems will consist of $\text{LPP}_3$ with the constraints:

$$x_1 \leq \lfloor \hat{x}_1 \rfloor = 3,$$

and:

$$x_1 \geq \lceil \hat{x}_1 \rceil = 4,$$

added. This gives the two new optimization problems:

$$\begin{aligned} \min_{x_1,x_2} \; & z = -x_1 - x_2 \\ \text{s.t. } & 17x_1 + 32x_2 \leq 136 \\ & 32x_1 + 15x_2 \leq 120 \\ & x_1, x_2 \geq 0 \\ & x_1 \geq 3 \\ & x_2 \leq 1 \\ & \boldsymbol{x_1 \leq 3}, \end{aligned}$$

and:

$$\begin{aligned} \min_{x_1,x_2} \; & z = -x_1 - x_2 \\ \text{s.t. } & 17x_1 + 32x_2 \leq 136 \\ & 32x_1 + 15x_2 \leq 120 \\ & x_1, x_2 \geq 0 \\ & x_1 \geq 3 \\ & x_2 \leq 1 \\ & \boldsymbol{x_1 \geq 4}, \end{aligned}$$

which we denote $\text{LPP}_5$ and $\text{LPP}_6$, respectively. We add these problems to $\Xi$, giving:

$$\Xi \leftarrow \Xi \cup \text{LPP}_5 \cup \text{LPP}_6 = \{\text{LPP}_1, \text{LPP}_4, \text{LPP}_5, \text{LPP}_6\}.$$

Figure 3.10 shows the tree-like structure of the problems in $\Xi$ at the end of the second Bound Updating and Branching step.

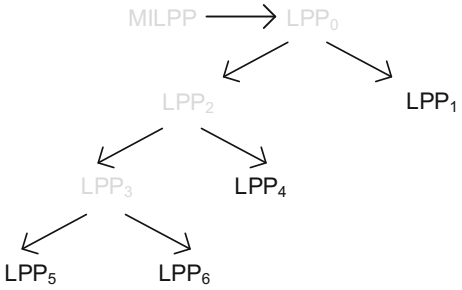


**Fig. 3.10** Second Bound Updating and Branching in the Branch-and-Bound Algorithm

We then go to the Optimality Check step. Finding that $\Xi \neq \emptyset$, we return to Step 2 and pick a new problem in $\Xi$ to solve.

In Step 2 we arbitrarily select $\text{LPP}_5$ to solve and find that it has as an optimal solution, $\hat{x} = (3, 1)^\top$, and corresponding objective-function value, $\hat{z} = -4$. We remove $\text{LPP}_5$ from $\Xi$, giving:

$$\Xi \leftarrow \{\text{LPP}_1, \text{LPP}_4, \cancel{\text{LPP}_5}, \text{LPP}_6\} = \{\text{LPP}_1, \text{LPP}_4, \text{LPP}_6\}.$$

Moving to the Bound Updating and Branching step, we find that $\hat{x}$ falls into Case 1, because it satisfies all of the integrality constraints of the original MILPP. We further have that $\hat{z} < z^u$. Thus, we can update the upper bound as:

$$z^u \leftarrow \hat{z} = -4,$$

and the best solution found thus far as:

$$x^b \leftarrow \hat{x} = (3, 1)^\top.$$

We also know that we do not have to add any other problems to $\Xi$ and can proceed to the next step.

We go to the Optimality Check step and find that $\Xi \neq \emptyset$. Thus, we must return to Step 2 and select another problem in $\Xi$ to solve. Figure 3.11 shows the new tree after $\Xi$ is updated by removing $\text{LPP}_5$.

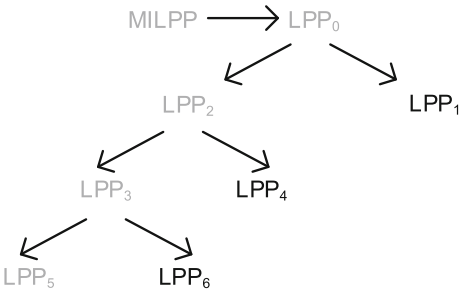


**Fig. 3.11** Third Bound Updating and Branching in the Branch-and-Bound Algorithm

In Step 2 we next select $\text{LPP}_6$ to solve, but find that it is infeasible. We then update $\Xi$ to:

$$\Xi \leftarrow \{\text{LPP}_1, \text{LPP}_4, \cancel{\text{LPP}_6}\} = \{\text{LPP}_1, \text{LPP}_4\}.$$

When we proceed to the Bound Updating and Branching step there is nothing to be done, because $\text{LPP}_6$ is infeasible. Thus, we proceed to the Optimality Check step and because $\Xi \neq \emptyset$ we return to Step 2. Figure 3.12 shows the updated tree after $\text{LPP}_6$ is removed.

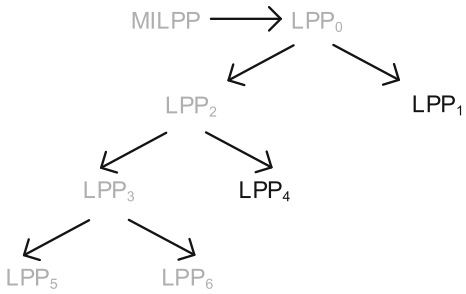


**Fig. 3.12** Fourth Bound Updating and Branching in the Branch-and-Bound Algorithm

In Step 2 we next select $\text{LPP}_4$ to solve and find that this problem is also infeasible. We update $\Xi$ to:

$$\Xi \leftarrow \{\text{LPP}_1, \cancel{\text{LPP}_4}\} = \{\text{LPP}_1\}.$$

As in the previous iteration, there is nothing to be done in the Bound Updating and Branching step, because $\text{LPP}_4$ is infeasible but the Optimality Check step tells us to return to Step 2 because $\Xi \neq \emptyset$. Figure 3.13 shows the updated tree after $\text{LPP}_6$ is removed.

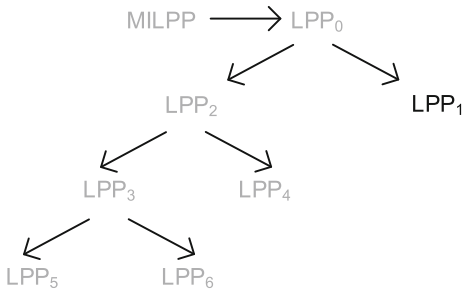


**Fig. 3.13** Fifth Bound Updating and Branching in the Branch-and-Bound Algorithm

We now solve $\text{LPP}_1$, which has $\hat{x} = (2, 3.187)^\top$ as an optimal solution and $\hat{z} = -5.187$ as its corresponding optimal objective-function value. We update $\Xi$ to:

$$\Xi \leftarrow \{\cancel{\text{LPP}_1}\} = \{\}.$$

We next proceed to the Bound Updating and Branching step and find that $\hat{x}$ falls into Case 3. We next find that $z^l \leq \hat{z} \leq z^u$ and $z^u < +\infty$, thus we can update the lower bound to:

$$z^l \leftarrow \hat{z} = -5.187.$$

Furthermore, because we have that $\hat{z} < z^u$, we must add two new problems to $\Xi$. $x_2$ is the only variable with a non-integer value in $\hat{x}$ that must be integer-valued in the original MILPP. Thus, these problems are formed by adding the constraints:

$$x_2 \leq \lfloor \hat{x}_2 \rfloor = 3,$$

and:

$$x_2 \geq \lceil \hat{x}_2 \rceil = 4,$$

to $\text{LPP}_1$, giving:

$$\begin{aligned} \min_{x_1,x_2} z &= -x_1 - x_2 \\ \text{s.t. } & 17x_1 + 32x_2 \leq 136 \\ & 32x_1 + 15x_2 \leq 120 \\ & x_1, x_2 \geq 0 \\ & x_1 \leq 2 \\ & \boldsymbol{x_2 \leq 3}, \end{aligned}$$

and:

$$\begin{aligned} \min_{x_1,x_2} z &= -x_1 - x_2 \\ \text{s.t. } & 17x_1 + 32x_2 \leq 136 \\ & 32x_1 + 15x_2 \leq 120 \\ & x_1, x_2 \geq 0 \\ & x_1 \leq 2 \\ & \boldsymbol{x_2 \geq 4}, \end{aligned}$$

as our two new problems, which we denote as $\text{LPP}_7$ and $\text{LPP}_8$, respectively. Adding these to $\Xi$ gives:

$$\Xi \leftarrow \Xi \cup \text{LPP}_7 \cup \text{LPP}_8 = \{\text{LPP}_7, \text{LPP}_8\},$$

and Figure 3.14 shows the updated tree. We next proceed to the Optimality Check. Because $\Xi \neq \emptyset$ we must return to Step 2.

In Step 2 we choose $\text{LPP}_7$ as the next problem to solve and find $\hat{x} = (2, 3)^\top$ to be its optimal solution and $\hat{z} = -5$ the corresponding objective-function value. We

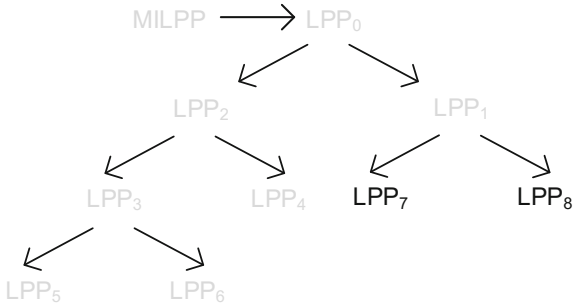


**Fig. 3.14** Sixth Bound Updating and Branching in the Branch-and-Bound Algorithm

update $\Xi$, giving:

$$\Xi \leftarrow \{\cancel{\text{LPP}_7}, \text{LPP}_8\} = \{\text{LPP}_8\}.$$

In the Bound Updating and Branching step we find that $\hat{x}$ falls into Case 1, because $\hat{x}$ satisfies all of the integrality constraints of the original MILPP. We further have that $\hat{z} < z^u$. Thus, we update the upper bound:

$$z^u \leftarrow \hat{z} = -5,$$

and the best feasible solution found thus far:

$$x^b \leftarrow \hat{x} = (2, 3)^\top.$$

We do not have to add any problems to $\Xi$ and thus proceed to the Optimality Check step. Because we still have $\Xi \neq \emptyset$ we return to Step 2. Figure 3.15 shows the new tree with $\text{LPP}_7$ removed.

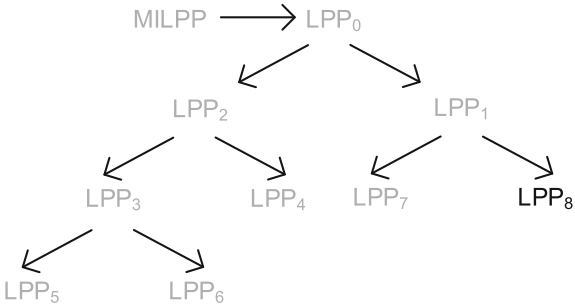


**Fig. 3.15** Seventh Bound Updating and Branching in the Branch-and-Bound Algorithm

Returning to Step 2, we solve $\text{LPP}_8$ (as it is the only problem remaining in $\Xi$) and find that it has $\hat{x} = (0.471, 4)^\top$ as an optimal solution with objective-function value $\hat{z} = -4.471$. Removing $\text{LPP}_8$ from $\Xi$ gives:

$$\Xi \leftarrow \{\cancel{\text{LPP}_8}\} = \{\}.$$

In the Bound Updating and Branching step we find that $\hat{x}$ falls into Case 3. However, because $\hat{z} > z^u$ we do not update the upper bound nor do we add any more problems to $\Xi$. Instead, we proceed to the Optimality Check and because we have $\Xi = \emptyset$, we terminate the Branch-and-Bound Algorithm. Moreover, because we have found a solution that is feasible in the original MILPP, $x^b = (2, 3)^\top$, we report this as the optimal solution of the original MILPP. Figure 3.16 shows the final tree of problems after the Branch-and-Bound Algorithm terminates. □

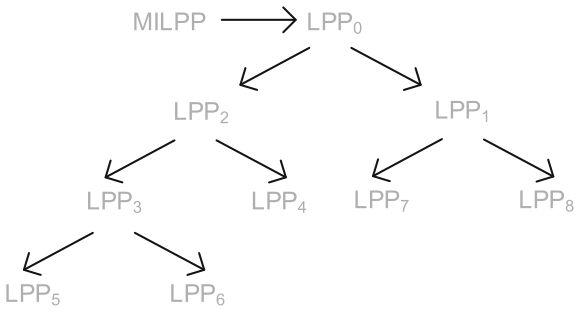


**Fig. 3.16** Final branch-and-bound tree

### *3.5.3 Rationale Behind Branch-and-Bound Algorithm*

The idea behind the Branch-and-Bound Algorithm is relatively straightforward. What the algorithm does is enumerates all of the different possible values that the integer variables can take in an MILPP. However, it does this in an intelligent fashion.

The heart of this enumeration is the Bound Updating and Branching step of the algorithm. Whenever a problem in $\Xi$ gives a solution in which an integer variable has a non-integer value, the algorithm creates two new problems in which that variable is fixed to be on the 'two sides' of that non-integer value. That is the logic behind adding constraints of the form:

$$x_i \leq \lfloor \hat{x}_i \rfloor,$$

and:

$$x_i \geq \lceil \hat{x}_i \rceil,$$

in Step 3. As we note in the discussion of Steps 1 and 3 in Sections 3.5.2.2 and 3.5.2.4, the constraints that are used to generate the new problems do not discard any feasible solutions to the original MILPP. They only work to generate LPPs that eventually generate solutions that are feasible in the original MILPP.

The 'intelligence' comes into play because we do not always add new problems after solving a problem in Step 2. For instance, we do not add new problems in Cases 1 and 2 of the Bound Updating and Branching step. The reason we do not in Case 1 is that the most recently solved LPP has already given a solution that is feasible in the original MILPP. It is true that solving a new problem in which an additional constraint is added to this LPP may give another feasible solution. However, any such feasible solutions will give worse objective-function values than the solution already found (this is a consequence of the Relaxation-Optimality Property because the most recently solved LPP is a relaxation of any LPP that we generate by adding constraints).

We do not add any new problems after Case 2 because if the most recently solved LPP is infeasible, then any LPP we generate by adding more constraints is also guaranteed to be infeasible.

Finally, in Case 3 of the Bound Updating and Branching step we do not add new problems if the most recently solved LPP gives an objective-function value, $\hat{z}$, that is greater than the current upper bound, $z^u$. The reason behind this is that if we do add new problems, their optimal solutions (whether they be feasible in the original MILPP or not) will have objective-function values that are worse (higher) than $\hat{z}$. This is because the most recently solved LPP is a relaxation of these problems that would have added constraints. If $z^u < \hat{z}$ that means we already have a solution that is feasible in the original MILPP and which gives a better objective-function value than any feasible solutions from these problems could give.

The intelligent enumeration of the Branch-and-Bound Algorithm can be seen in the amount of work involved in solving the Photovoltaic Panel-Repair Problem in Example 3.1. The Branch-and-Bound Algorithm requires us to solve nine LPPs (when we count the first linear relaxation). As a result, we are implicitly examining nine possible solutions to the MILPP. Figure 3.1 shows that this problem has 15 feasible solutions. Because of its intelligence, we typically only have to examine a subset of solutions when solving a MILPP using the Branch-and-Bound Algorithm.

### *3.5.4 Branch-and-Bound Algorithm*

We now give a more detailed outline of the Branch-and-Bound Algorithm. Lines 2–16 combine the Initialization and Initial Branching steps, which are discussed in Sections 3.5.2.1 and 3.5.2.2. First, Lines 2–4 initialize $z^l$, $z^u$, and $\Xi$. Next the linear relaxation is solved in Line 5. Lines 6–9 terminate the algorithm if the linear relaxation shows the original MILPP to be infeasible or gives an optimal solution to the MILPP.

**Branch-and-Bound Algorithm**

1: **procedure** BRANCH AND BOUND
2: $z^l \leftarrow -\infty$
3: $z^u \leftarrow +\infty$
4: $\Xi \leftarrow \emptyset$
5: solve $\text{LPP}_0$ ▷ $x^0$, $z^0$ denote optimal solution and objective-function value
6: **if** $\text{LPP}_0$ is infeasible **then**
7: **stop**, original MILPP is infeasible
8: **else if** $x^0$ satisfies integrality constraints of original MILPP **then**
9: **stop**, $x^0$ is optimal in original MILPP
10: **else**
11: $z^l \leftarrow z^0$
12: select a variable, $x_i$, to branch on
13: generate $\text{LPP}^-$, which is $\text{LPP}_0$ with constraint $x_i \leq \lfloor x_i^0 \rfloor$ added
14: generate $\text{LPP}^+$, which is $\text{LPP}_0$ with constraint $x_i \geq \lceil x_i^0 \rceil$ added
15: $\Xi \leftarrow \Xi \cup \text{LPP}^- \cup \text{LPP}^+$
16: **end if**
17: **repeat**
18: select a problem in $\Xi$ ▷ denote the selected problem 'LPP'
19: remove LPP from $\Xi$
20: solve LPP ▷ $\hat{x}$, $\hat{z}$ denote optimal solution and objective-function value
21: **if** $\hat{x}$ satisfies integrality constraints of original MILPP **then**
22: **if** $\hat{z} < z_u$ **then**
23: $z^u \leftarrow \hat{z}$
24: $x^b \leftarrow \hat{x}$
25: **end if**
26: **else if** LPP is feasible **then**
27: **if** $z^l < \hat{z} \leq z^u$ and $z^u < +\infty$ **then**
28: $z^l \leftarrow \hat{z}$
29: **end if**
30: **if** $\hat{z} < z^u$ **then**
31: select a variable, $x_i$, to branch on
32: generate $\text{LPP}^-$, which is LPP with constraint $x_i \leq \lfloor x_i^0 \rfloor$ added
33: generate $\text{LPP}^+$, which is LPP with constraint $x_i \geq \lceil x_i^0 \rceil$ added
34: $\Xi \leftarrow \Xi \cup \text{LPP}^- \cup \text{LPP}^+$
35: **end if**
36: **end if**
37: **until** $\Xi = \emptyset$
38: **end procedure**

Otherwise, the lower bound is updated in Line 11. Note that if the linear relaxation is unbounded the value of $z^l$ does not actually change, because $z^0 = -\infty$. In Line 12 we select one variable, which we denote $x_i$, to branch on. The variable to branch on can be any variable that has a non-integer value in $x^0$ but which is constrained to take on an integer value in the original MILPP. We then generate the two new LPPs, which have the same objective function and constraints as the linear relaxation, but each have one new constraint, which are:

$$x_i \leq \lfloor x_i^0 \rfloor,$$

and:

$$x_i \geq \lceil x_i^0 \rceil.$$

Lines 17–37 are the main iterative loop of the algorithm. We begin by selecting a problem, which we denote LPP, in $\Xi$, removing LPP from $\Xi$, and then solving LPP in Lines 18–20. We let $\hat{x}$ and $\hat{z}$ denote the optimal solution and objective-function value of LPP. Lines 21–36 are the Bound Updating and Branching process, which is outlined in Section 3.5.2.4. Lines 21–25 handle Case 1 of this process, where $\hat{x}$ satisfies the integrality constraints of the original MILPP. If so, we update the upper bound and the best solution found in Lines 23 and 24, so long as $\hat{z} < z^u$. We do not have to branch (*i.e.*, add new problems to $\Xi$) in this case.

Lines 26–36 handle Case 3 in Section 3.5.2.4. If LPP is feasible but $\hat{x}$ is not feasible in the original MILPP, we then update the lower bound in Line 28 if $z^l < \hat{z} \leq z^u$ and $z^u < +\infty$. Also, if $\hat{z} < z^u$ we select another variable to branch on and generate the two LPPs that are added to $\Xi$ in Lines 31–34.

Note that Lines 17–37 do not explicitly discuss cases in which LPP is infeasible or unbounded. If LPP is infeasible, there is no bound updating to be done and no new LPPs to be generated and added to $\Xi$. Thus, we do not do anything after solving LPP in that iteration. If LPP is unbounded, then we generate new optimization problems that are added to $\Xi$ in Lines 31–34.

Line 37 is the Optimality Check. This is because we continue the iterative loop until $\Xi = \emptyset$, which is the test conducted in Section 3.5.2.5.

### 3.5.5 *Practical Considerations*

There are three important practical issues to consider when applying the Branch-and-Bound Algorithm. The first involves which variable to branch on (*i.e.*, which variable to use when adding constraints to the new problems being added to $\Xi$) in the Initial Branching and the Bound Updating and Branching steps of the algorithm. One may be tempted to wonder which is the 'best' variable to branch on, in the sense of obtaining the optimal solution to the original MILPP as quickly as possible. Unfortunately, the answer to this question is usually highly problem-dependent and complex relationships between different variables can make it difficult to ascertain this *a priori*. Thus, no general rules are available. Most MILPP solvers employ heuristic rules, and the cost of a solver package is often tied to how much research and sophistication is involved in the rules implemented.

A second question is what order to process the problems in $\Xi$ in Step 2 of the algorithm. There is, again, no general rule that applies to all problems because the efficiency of solving a MILPP is highly dependent on the structure of the objective function and constraints. Two heuristic rules that are commonly employed are known as **depth-first** and **breadth-first** strategies. A depth-first strategy, which we employ

in Example 3.1, goes deep down the branches of the tree as quickly as possible. This is seen in the strategy that we employ because we solve $\text{LPP}_2$, followed by $\text{LPP}_3$ and $\text{LPP}_5$. A breadth-first strategy would, instead, stay at the top levels of the tree first before going deeper. Applying such a strategy to Example 3.1 would see us solve $\text{LPP}_2$ and $\text{LPP}_1$ before moving on the $\text{LPP}_3$, $\text{LPP}_4$, $\text{LPP}_7$, and $\text{LPP}_8$ and only then going on to $\text{LPP}_5$ and $\text{LPP}_6$.

The primary benefit of a depth-first strategy is that it quickly produces problems with many constraints that are either infeasible or give feasible solutions to the original MILPP. This allows us to tighten the upper and lower bounds and find feasible solutions relatively quickly. On the other hand, a breadth-first strategy allows us to solve LPPs that are very similar to each other (they only have a small number of constraints that differ from one another). This often allows us to solve the LPPs more quickly. That being said, there is no general rule as to which of the two strategies works most efficiently. Indeed, many solvers employ a combination of the two approaches.

A third issue, which is raised in Section 3.5.1, is that we often do not solve an MILPP to complete optimality. That means, we may terminate the Branch-and-Bound Algorithm before every problem in $\Xi$ is solved. This process is normally governed by the upper and lower bounds. If these bounds are sufficiently close to one another, we may terminate the algorithm because we have a solution that is 'close enough' to optimal. For instance, in Example 3.1 we find the solution $x = (3, 1)^\top$, which is not optimal after, solving $\text{LPP}_5$. We also know, after solving $\text{LPP}_5$, that $z^l = -5.347$ and $z^u = -4$. Thus, we know that the solution we have is at most:

$$\left| \frac{z^u - z^l}{z^l} \right| = 0.25,$$

or 25% away from the optimal solution. In many cases, we may not be sufficiently comfortable to use the feasible solution that we have at hand. However, if we find a solution and know that it is at most 0.01% away from optimal, we may be happy with that. Indeed, by default most MILPP solvers do not solve MILPPs to complete optimality but stop once this so-called **optimality gap** is sufficiently small.

## 3.6 Solving Pure-Integer Linear Optimization Problems Using Cutting Planes

This section outlines a fundamentally different way of solving a mixed-integer linear optimization problem. Although the technique that we outline here can be applied to generic MILPPs, we focus our attention on pure-integer linear optimization problems. Interested readers may consult other more advanced texts [13] that discuss the generalization of this technique to generic MILPPs.

The idea of this algorithm is relatively straightforward and relies on some simple properties of MILPPs and LPPs. We first know that if the integrality constraints of a MILPP are relaxed, the resulting linear relaxation is an LPP. Thus, we know that an optimal solution to the linear relaxation will have the properties discussed in Section 2.3. More specifically, we argue in Section 2.3.1 that the feasible region of every LPP has at least one extreme point or corner that is optimal.

To see why this observation is important, examine the feasible region of the linear relaxation of the Photovoltaic Panel-Repair Problem, which is shown in Figure 3.17. We know from the properties of linear optimization problems that when we solve this linear relaxation, one of the four extreme points of the linear relaxation, which are highlighted as red squares in Figure 3.17, will be the solution given by the Simplex method. Unfortunately, only one of these extreme points:

$$(x_1, x_2) = (0, 0)^\top,$$

is a solution that is feasible in the original MILPP, and it is not an optimal solution to the MILPP.

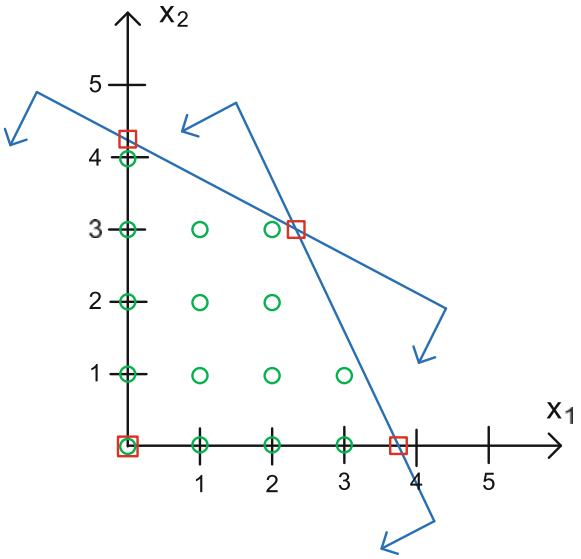


**Fig. 3.17** Geometrical representation of the feasible region of the linear relaxation of the Photovoltaic Panel-Repair Problem

The idea that we explore in this section is to solve the MILPP by adding what are known as **cutting planes** [8]. A cutting plane is simply a constraint that cuts off solutions that are feasible in the linear relaxation but are not feasible in the original MILPP. Figure 3.18 shows the feasible region of the linear relaxation of the original MILPP when these cutting planes are added. Note that the additional constraints make the five extreme points of the feasible region (which are highlighted with red squares) coincide with points where $x$ is integer-valued (and, thus, feasible in the

original MILPP). The benefit of these cutting planes is that once we solve the linear relaxation, the Simplex method gives us one of the extreme points, which will have integer values for $x$, as an optimal solution. The important observation about cutting planes is that they change (and reduce) the feasible region of the linear relaxation. However, they do not change the feasible region of the original MILPP.

In practice, we do not have all of the cutting planes needed to solve a MILPP *a priori*. Instead, we generate the cutting planes in an iterative algorithm by solving the linear relaxation and finding constraints that cut off non-integer solutions. After a sufficient number of iterations, and adding a sufficient number of these constraints, we have a linear relaxation that gives an integer-valued solution that is feasible in the original MILPP.

We proceed in this section by first outlining how to generate cuts from a non-integer solution obtained from solving the linear relaxation of a pure-integer linear optimization problem. We then outline the iterative algorithm to generate cutting planes and demonstrate its use with an example.

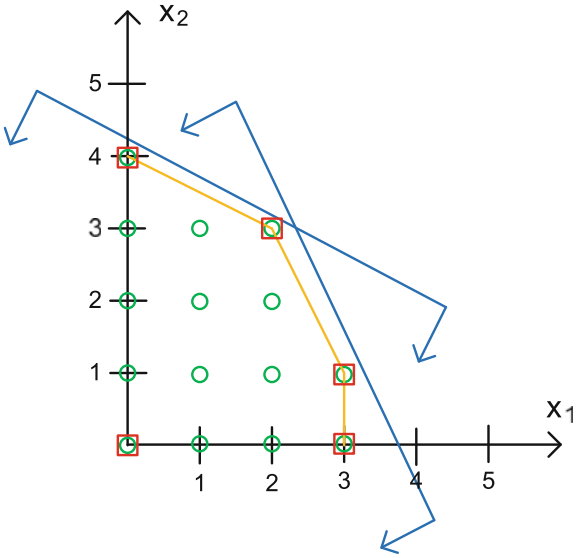


**Fig. 3.18** Geometrical representation of the feasible region of the linear relaxation of the Photovoltaic Panel-Repair Problem with cutting planes added

## *3.6.1 Generating Cutting Planes*

To derive a cutting plane, let us suppose that we solve the linear relaxation of a PILPP and obtain a solution, $\hat{x}$, which does not satisfy all of the integrality restrictions of the original PILPP. We further assume that the linear relaxation is converted to standard form (2.14)–(2.16):

$$\begin{aligned} \min_x \; & c^\top x \\ \text{s.t. } & Ax = b \\ & x \geq 0, \end{aligned}$$

which is introduced in Section 2.2.2.1. This form allows us to analyze the structural equality constraints by partitioning $x$ into basic and non-basic variables (*cf.* Section 2.4.1 for further details). This partition allows us to write the equality constraints as:

$$\begin{bmatrix} B & N \end{bmatrix} \begin{pmatrix} x_B \\ x_N \end{pmatrix} = b, \tag{3.44}$$

where we partition $A$ into:

$$A = \begin{bmatrix} B & N \end{bmatrix},$$

and the $B$ submatrix is full-rank and we partition $x$ into:

$$x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}.$$

Equation (3.44) can be rewritten as:

$$Bx_B + Nx_N = b,$$

or, because $B$ is full-rank, as:

$$x_B + B^{-1}Nx_N = B^{-1}b.$$

This can further be simplified as:

$$x_B + \tilde{N}x_N = \tilde{b}, \tag{3.45}$$

where:

$$\tilde{N} = B^{-1}N, \tag{3.46}$$

and:

$$\tilde{b} = B^{-1}b. \tag{3.47}$$

We next divide the values of $\tilde{N}$ and $\tilde{b}$ into their integer and non-integer components. More specifically, define:

$$\tilde{N}^I = \left\lfloor \tilde{N} \right\rfloor,$$

$$\tilde{N}^F = \tilde{N} - \tilde{N}^I,$$

$$\tilde{b}^I = \left\lfloor \tilde{b} \right\rfloor,$$

and:

$$\tilde{b}^F = \tilde{b} - \tilde{b}^I.$$

We can note a few properties of $\tilde{N}$, $\tilde{b}$, $\tilde{N}^I$, $\tilde{N}^F$, $\tilde{b}^I$, and $\tilde{b}^F$. First, we clearly have that:

$$\tilde{N} = \tilde{N}^I + \tilde{N}^F,$$

and:

$$\tilde{b} = \tilde{b}^I + \tilde{b}^F,$$

from the definitions of $\tilde{N}^I$, $\tilde{N}^F$, $\tilde{b}^I$, and $\tilde{b}^F$. Next, we know that $\tilde{N}^I$ and $\tilde{b}^I$ are integer-valued, because they are defined as the floors of $\tilde{N}$ and $\tilde{b}$, respectively. Finally, we have that $\tilde{N}^F$ and $\tilde{b}^F$ are non-integer-valued, non-negative, and less than 1, because they are defined as the difference between each of $\tilde{N}$ and $\tilde{b}$ and their floors.

Thus, we can rewrite (3.45) as:

$$x_B + (\tilde{N}^I + \tilde{N}^F)x_N = \tilde{b}^I + \tilde{b}^F,$$

or, by rearranging terms, as:

$$x_B + \tilde{N}^I x_N - \tilde{b}^I = \tilde{b}^F - \tilde{N}^F x_N. \tag{3.48}$$

We next consider a basic variable, which we denote $x_{B,i}$, which has a non-integer value in $\hat{x}$. Note that because the original PILPP requires all of the variables to be integer-valued, the value of $\hat{x}_{B,i}$ is not feasible in the PILPP. We can define the value of $x_{B,i}$ from (3.48) as:

$$x_{B,i} + \sum_{j=1}^{m} \tilde{N}^I_{i,j} x_{N,j} - \tilde{b}^I_i = \tilde{b}^F_i - \sum_{j=1}^{m} \tilde{N}^F_{i,j} x_{N,j}, \tag{3.49}$$

where $m$ is the number of structural equality constraints when the linear relaxation is written in standard form.

Note that the left-hand side of (3.49) is, by definition, integer-valued in the original PILPP. To see why, note that all of the $x$'s are restricted to be integer-valued in the original PILPP. Moreover, the coefficients $\tilde{N}^I_{i,j}$ and the constant $\tilde{b}^I_i$ are defined to be integer-valued as well. Thus, for (3.49) to hold, the right-hand side must be integer-valued as well.

Let us next examine the right-hand side of (3.49). We can first argue that $\tilde{b}^F_i > 0$. The reason for this is that we are focusing our attention on a basic variable that is not integer-valued in $\hat{x}$. Moreover, we know from the discussion in Section 2.4.2 that the values of basic variables are equal to $\tilde{b}$, because the non-basic variables are fixed equal to zero in the Simplex method. Thus, because $x_{B,i} = \tilde{b}_i$, if $\hat{x}_{B,i}$ is not

integer-valued, then $\tilde{b}_i$ must have a strictly positive non-integer component, $\tilde{b}_i^F$. We further know that the second term on the right-hand side of (3.49):

$$\sum_{j=1}^{m} \tilde{N}_{i,j}^F x_{N,j},$$

is non-negative, because the coefficients, $\tilde{N}_{i,j}^F$, and the variables, $x_{N,j}$, are all non-negative. Because $\tilde{b}^F$ is defined as the non-integer component of $\tilde{b}$, it is by definition strictly less than 1. Thus, we can conclude that the right-hand side of (3.49) is an integer that is less than or equal to zero, or that:

$$\tilde{b}_i^F - \sum_{j=1}^{m} \tilde{N}_{i,j}^F x_{N,j} \leq 0,$$

which can also be written as:

$$\sum_{j=1}^{m} \tilde{N}_{i,j}^F x_{N,j} - \tilde{b}_i^F \geq 0. \tag{3.50}$$

Inequality (3.50) is the cutting plane that is generated by the solution, $\hat{x}$. This type of cutting plane is often referred to as a Gomory cut, as Gomory introduced this method of solving MILPPs [8]. Cuts of this form are used in the iterative algorithm that is outlined in the next section.

## *3.6.2 Outline of Cutting-Plane Algorithm*

Building off of the derivation of the Gomory cut given in Section 3.6.1, we now outline the major steps of the Cutting-Plane Algorithm.

### 3.6.2.1 Step 0: Initialization

We begin the Cutting-Plane Algorithm by first solving the linear relaxation of the original PILPP. When we solve the linear relaxation, one of the following three outcomes is possible.

1. The linear relaxation may be infeasible. If so, then based on the Relaxation-Feasibility Property, the original PILPP is infeasible as well. As such, we terminate the Cutting-Plane Algorithm and report that the original PILPP is infeasible.
2. Solving the linear relaxation may give a solution that satisfies all of the integrality constraints of the original PILPP. If so, then based on the Relaxation-Optimality

Corollary, the optimal solution of the linear relaxation is also optimal in the original PILPP. Thus, we terminate the algorithm and report the solution found as being optimal.
3. If the solution to the linear relaxation does not fit into the first two cases (*i.e.*, we obtain a solution with non-integer values for some of the variables or the linear relaxation is unbounded), then we proceed with the Cutting-Plane Algorithm. Let $\hat{x}$ denote the optimal solution found from solving the linear relaxation.

#### 3.6.2.2 Step 1: Cut Generation

Select a variable, which we denote $x_i$, that has a non-integer value in $\hat{x}$. In theory, any non-integer-valued variable can be used. In practice, it is often beneficial to select the one that has the largest non-integer component when $\tilde{b}$ is decomposed into $\tilde{b}^I$ and $\tilde{b}^F$. Generate cut (3.50) for the chosen variable.

#### 3.6.2.3 Step 2: Solving

Add the cut generated in Step 1 to the most recently solved LPP and solve the resulting LPP.

#### 3.6.2.4 Step 3: Optimality Check

When we solve the LPP in Step 2 there are three possible outcomes.

1. The LPP may be infeasible. If so, then the original PILPP is infeasible as well. As such, we terminate the Cutting-Plane Algorithm and report that the original PILPP is infeasible.
2. Solving the LPP may give a solution that satisfies all of the integrality constraints of the original PILPP. If so, then the optimal solution of the LPP is also optimal in the original PILPP. Thus, we terminate the algorithm and report the solution found as being optimal.
3. If the solution to the LPP does not fit into these two cases (*i.e.*, we obtain a solution with non-integer values for some of the variables or the LPP is unbounded) then we continue the algorithm by returning to Step 1. Let $\hat{x}$ denote the optimal solution found from solving the LPP.

*Example 3.2* Consider the following variant of the Photovoltaic Panel-Repair Problem from Section 3.1.1. A type-A repair unit is now 10% more effective than a type-B unit. Moreover, each type-A unit has a mass of 2 kg and occupies 7 $m^3$ of space while a type-B unit has a mass of 1 kg and occupies 8 $m^3$. The shuttle can now carry at most 6 kg of repair units and has at most 28 $m^3$ of space available in its cargo bay. The payload specialists must determine how many units of each type to carry aboard

the spacecraft to maximize the effectiveness-weighted number of repair units sent to the spacecraft.

To formulate this problem we let $x_1$ and $x_2$, respectively, denote the number of type-A and -B repair units put into the spacecraft. The PILPP is then:

$$\begin{aligned}\min_{x_1,x_2} z &= -\frac{11}{10}x_1 - x_2\\ \text{s.t. } 2x_1 + x_2 &\leq 6\\ 7x_1 + 8x_2 &\leq 28\\ x_1, x_2 &\geq 0\\ x_1, x_2 &\in \mathbb{Z},\end{aligned}$$

when the objective function is converted into a minimization.

To solve this problem using the Cutting-Planes Algorithm, we first convert the linear relaxation:

$$\begin{aligned}\min_{x_1,x_2} z &= -\frac{11}{10}x_1 - x_2\\ \text{s.t. } 2x_1 + x_2 &\leq 6\\ 7x_1 + 8x_2 &\leq 28\\ x_1, x_2 &\geq 0,\end{aligned}$$

into standard form:

$$\begin{aligned}\min_{x_1,x_2,x_3,x_4} z &= -\frac{11}{10}x_1 - x_2\\ \text{s.t. } 2x_1 + x_2 + x_3 &= 6\\ 7x_1 + 8x_2 + x_4 &= 28\\ x_1, x_2, x_3, x_4 &\geq 0,\end{aligned}$$

by adding two new slack variables, $x_3$ and $x_4$. Solving this problem gives $\hat{x} = (20/9, 14/9, 0, 0)^\top$. Because this solution does not satisfy all of the integrality constraints of the original PILPP, we add a new cut.

To do so, we first recall that we have:

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 7 & 8 & 0 & 1 \end{bmatrix},$$

and:

$$b = \begin{pmatrix} 6 \\ 28 \end{pmatrix}.$$

Because $x_1$ and $x_2$ are basic variables and $x_3$ and $x_4$ are non-basic variables, we know that the basis matrix will have the first two columns of $A$:

$$B = \begin{bmatrix} 2 & 1 \\ 7 & 8 \end{bmatrix},$$

and the $N$ matrix will have the remaining columns:

$$N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Using (3.46) and (3.47) we have:

$$\tilde{N} = \begin{bmatrix} 8/9 & -1/9 \\ -7/9 & 2/9 \end{bmatrix},$$

and:

$$\tilde{b} = \begin{pmatrix} 20/9 \\ 14/9 \end{pmatrix}.$$

We can decompose these two into their integer and non-integer parts:

$$\tilde{N}^I = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix},$$

$$\tilde{N}^F = \begin{bmatrix} 8/9 & 8/9 \\ 2/9 & 2/9 \end{bmatrix},$$

$$\tilde{b}^I = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

and:

$$\tilde{b}^F = \begin{pmatrix} 2/9 \\ 5/9 \end{pmatrix}.$$

We now select either of $x_1$ or $x_2$ to generate a cutting plane with. Because the non-integer components of $\hat{x}$ are given by $\tilde{b}^F$ and $\tilde{b}^F_2 > \tilde{b}^F_1$, we generate a cut using $x_2$. From (3.50) this cut is given by:

$$\sum_{j=1}^{m} \tilde{N}^F_{2,j} x_{N,j} - \tilde{b}^F_2 \geq 0,$$

or by:

$$\frac{2}{9}x_3 + \frac{2}{9}x_4 - \frac{5}{9} \geq 0,$$

when we substitute in the values of $\tilde{N}^F$ and $\tilde{b}^F$. We can further simplify this cut to:

$$2x_3 + 2x_4 - 5 \geq 0.$$

Adding this cut to the standard form of the linear relaxation of the PILPP gives:

$$\begin{aligned} \min_{x_1,x_2,x_3,x_4} \; & z = -\frac{11}{10}x_1 - x_2 \\ \text{s.t. } & 2x_1 + x_2 + x_3 = 6 \\ & 7x_1 + 8x_2 + x_4 = 28 \\ & 2x_3 + 2x_4 - 5 \geq 0 \\ & x_1, x_2, x_3, x_4 \geq 0, \end{aligned}$$

which we transform into the standard-form problem:

$$\begin{aligned} \min_{x_1,x_2,x_3,x_4,x_5} \; & z = -\frac{11}{10}x_1 - x_2 \\ \text{s.t. } & 2x_1 + x_2 + x_3 = 6 \\ & 7x_1 + 8x_2 + x_4 = 28 \\ & 2x_3 + 2x_4 - x_5 = 5 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0, \end{aligned}$$

by adding the surplus variable, $x_5$, to the problem. Solving this problem gives $\hat{x} = (5/2, 1, 0, 5/2, 0)^\top$. This solution does not satisfy the integrality constraints of the original PILPP, because $\hat{x}_1 = 5/2$.

Thus, we must generate a new cutting plane. To do so, we first note that we now have:

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 7 & 8 & 0 & 1 & 0 \\ 0 & 0 & 2 & 2 & -1 \end{bmatrix},$$

and:

$$b = \begin{pmatrix} 6 \\ 28 \\ 5 \end{pmatrix}.$$

Because $x_1$, $x_2$, and $x_4$ are basic variables we have:

$$B = \begin{bmatrix} 2 & 1 & 0 \\ 7 & 8 & 1 \\ 0 & 0 & 2 \end{bmatrix},$$

and:

$$N = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 2 & -1 \end{bmatrix},$$

and from (3.46) and (3.47) we have:

$$\tilde{N} = \begin{bmatrix} 1 & -1/18 \\ -1 & 1/9 \\ 1 & -1/2 \end{bmatrix},$$

and:

$$\tilde{b} = \begin{pmatrix} 5/2 \\ 1 \\ 5/2 \end{pmatrix}.$$

Decomposing these into their integer and non-integer parts gives:

$$\tilde{N}^I = \begin{bmatrix} 1 & -1 \\ -1 & 0 \\ 1 & -1 \end{bmatrix},$$

$$\tilde{N}^F = \begin{bmatrix} 0 & 17/18 \\ 0 & 1/9 \\ 0 & 1/2 \end{bmatrix},$$

$$\tilde{b}^I = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

and:

$$\tilde{b}^F = \begin{pmatrix} 1/2 \\ 0 \\ 1/2 \end{pmatrix}.$$

Note that the value of $\hat{x}_4 = 5/2$ does not affect the feasibility of this solution in the original PILPP, because the original PILPP only constraints $x_1$ and $x_2$ to be integer-valued. However, if we examine the structural constraint:

$$7x_1 + 8x_2 + x_4 = 28,$$

it is clear that if $x_4$ takes on a non-integer value, then at least one of $x_1$ or $x_2$ will also take on a non-integer value. Thus, we have the option of generating a cut using either of $x_1$ or $x_4$. If we select $x_4$, the new cutting plane will be:

$$\frac{1}{2}x_5 - \frac{1}{2} \geq 0,$$

or:

$$x_5 - 1 \geq 0,$$

when simplified. Adding this constraint to the current LPP gives:

$$\begin{aligned} \min_{x_1,x_2,x_3,x_4,x_5} z = & -\frac{11}{10}x_1 - x_2 \\ \text{s.t. } & 2x_1 + x_2 + x_3 = 6 \\ & 7x_1 + 8x_2 + x_4 = 28 \\ & 2x_3 + 2x_4 - x_5 = 5 \\ & x_5 - 1 \geq 0 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0, \end{aligned}$$

which is:

$$\begin{aligned} \min_{x_1,x_2,x_3,x_4,x_5,x_6} z = & -\frac{11}{10}x_1 - x_2 \\ \text{s.t. } & 2x_1 + x_2 + x_3 = 6 \\ & 7x_1 + 8x_2 + x_4 = 28 \\ & 2x_3 + 2x_4 - x_5 = 5 \\ & x_5 - x_6 = 1 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0, \end{aligned}$$

in standard form.

Solving the new LPP gives $x = (23/9, 8/9, 0, 3, 1, 0)^\top$. Thus, we must generate a new cut. To do so, we note that we now have:

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 & 0 \\ 7 & 8 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix},$$

$$b = \begin{pmatrix} 6 \\ 28 \\ 5 \\ 1 \end{pmatrix},$$

$$B = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 7 & 8 & 1 & 0 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

and:

$$N = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 2 & 0 \\ 0 & -1 \end{bmatrix}.$$

From (3.46) and (3.47) we also have:

$$\tilde{N} = \begin{bmatrix} 1 & -1/18 \\ -1 & 1/9 \\ 1 & -1/2 \\ 0 & -1 \end{bmatrix},$$

and:

$$\tilde{b} = \begin{pmatrix} 25/9 \\ 4/9 \\ 5 \\ 5 \end{pmatrix},$$

which are decomposed as:

$$\tilde{N}^I = \begin{bmatrix} 1 & -1 \\ -1 & 0 \\ 1 & -1 \\ 0 & -1 \end{bmatrix},$$

$$\tilde{N}^F = \begin{bmatrix} 0 & 17/18 \\ 0 & 1/9 \\ 0 & 1/2 \\ 0 & 0 \end{bmatrix},$$

$$\tilde{b}^I = \begin{pmatrix} 2 \\ 0 \\ 5 \\ 5 \end{pmatrix},$$

and:

$$\tilde{b}^F = \begin{pmatrix} 7/9 \\ 4/9 \\ 0 \\ 0 \end{pmatrix}.$$

Generating a cut with $x_2$ gives:

$$\frac{1}{9}x_6 - \frac{4}{9} \geq 0,$$

or:

$$x_6 - 4 \geq 0.$$

Adding this to our current LPP gives:

$$\begin{aligned}
\min_{x_1,x_2,x_3,x_4,x_5,x_6} \quad & z = -\frac{11}{10}x_1 - x_2 \\
\text{s.t. } & 2x_1 + x_2 + x_3 = 6 \\
& 7x_1 + 8x_2 + x_4 = 28 \\
& 2x_3 + 2x_4 - x_5 = 5 \\
& x_5 - x_6 = 1 \\
& x_6 - 4 \geq 0 \\
& x_1, x_2, x_3, x_4, x_5, x_6 \geq 0,
\end{aligned}$$

which is:

$$\begin{aligned}
\min_{x_1,x_2,x_3,x_4,x_5,x_6,x_7} \quad & z = -\frac{11}{10}x_1 - x_2 \\
\text{s.t. } & 2x_1 + x_2 + x_3 = 6 \\
& 7x_1 + 8x_2 + x_4 = 28 \\
& 2x_3 + 2x_4 - x_5 = 5 \\
& x_5 - x_6 = 1 \\
& x_6 - x_7 = 4 \\
& x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0,
\end{aligned}$$

in standard form. The optimal solution to this LPP is $x = (3, 0, 0, 7, 9, 4)^\top$, which satisfies all of the integrality constraints of the original PILPP. Thus, we terminate the Cutting-Plane Algorithm and report $(x_1, x_2) = (3, 0)$ as an optimal solution. □

### *3.6.3 Cutting-Plane Algorithm*

We now give a more detailed outline of the Cutting-Plane Algorithm. Lines 2–5 initialize the algorithm. We have three flags that indicate if the algorithm should terminate because the original PILPP is found to be infeasible or unbounded or if we find an optimal solution to the original PILPP. These three flags are initially set to zero in Lines 2–4. The algorithm also tracks a 'current LPP,' the constraints of

which are updated as new cutting planes are generated. In Line 5, the current LPP is set equal to the linear relaxation of the PILPP.

**Cutting-Plane Algorithm**

```
1: procedure CUTTING PLANE
2:     τ^I ← 0
3:     τ^U ← 0
4:     τ^S ← 0
5:     'current LPP' ← 'linear relaxation of PILPP'        ▷ x̂ denotes optimal solution
6:     repeat
7:         solve current LPP
8:         if most recently solved LPP is infeasible then
9:             τ^I ← 1
10:        else if x̂ satisfies integrality constraints of original PILPP then
11:            if objective function is unbounded then
12:                τ^U ← 1
13:            else
14:                τ^S ← 1
15:            end if
16:        else
17:            determine B and N from final tableau of most recently solved LPP
18:            Ñ ← B^{-1}N
19:            b̃ ← B^{-1}b
20:            Ñ^I ← ⌊Ñ⌋
21:            Ñ^F ← Ñ − Ñ^I
22:            b̃^I ← ⌊b̃⌋
23:            b̃^F ← b̃ − b̃^I
24:            select a variable, x_i, to add a cut for
25:            add constraint Σ_{j=1}^{m} Ñ^F_{i,j} x_{N,j} − b̃^F_i ≥ 0 to most recently solved LPP
26:        end if
27:    until τ^I = 1 or τ^U = 1 or τ^S = 1
28:    if τ^I = 1 then
29:        original PILPP is infeasible
30:    else if τ^U = 1 then
31:        original PILPP is unbounded
32:    else
33:        x̂ is optimal in original PILPP
34:    end if
35: end procedure
```

Lines 6–27 are the main iterative loop of the algorithm. The current LPP is solved in Line 7 and one of four things happens depending on the solution found. If the current LPP is infeasible, then the flag $\tau^I$ is set equal to 1 in Lines 8 and 9 to indicate that the original PILPP is found to be infeasible. Next, if the optimal solution to the current LPP satisfies the integrality constraints of the original PILPP and the objective function is bounded, then we have found an optimal solution to the original PILPP and set the flag $\tau^S$ equal to 1 in Lines 13 and 14. Otherwise, if the objective

function is unbounded, then the original PILPP is unbounded as well and the flag $\tau^U$ is set equal to 1 in Lines 11 and 12. Note that this case only applies if we have an integer-valued solution that gives an unbounded objective function. If the LPP solved in Line 7 is unbounded but we cannot guarantee that there is an integer-valued solution that is unbounded, this falls into the final case in which a new cut is added to the current problem. In the final case, in which the LPP is feasible but we do not find a solution that satisfies the integrality constraints of the original PILPP (or the LPP is unbounded but we cannot guarantee that there is an integer-valued solution that is unbounded), we add a new cut to the current problem in Lines 17–25.

This iterative process repeats until we terminate the algorithm in Line 27 for one of the three reasons (optimality, infeasibility, or unboundedness). Lines 28–34 determine what to report, based on which of $\tau^I$, $\tau^U$, or $\tau^S$ is equal to 1 when the algorithm terminates.

## 3.7 Final Remarks

Current techniques to efficiently solve large-scale MILPPs (*e.g.*, those implemented in CPLEX [10] or GUROBI [9]) combine the branch-and-bound algorithm with cutting-plane methods to reduce the feasible region of the LPPs that must be solved while also exploring the branch-and-bound tree. These hybrid techniques pay particular attention to adding cutting planes to the original linear relaxation of the MILPP before branching begins. The reason for this is that cutting planes added to the linear relaxation are carried through in all of the subsequent LPPs that are solved and tend to improve the quality of the solutions found. This, in turn, increases solution efficiency. Commercial MILPP solvers also incorporate numerous heuristics to determine which variable to branch on and which LPP in $\Xi$ to solve in each iteration of the Branch-and-Bound Algorithm. Bixby [2] provides an excellent overview of the evolution of MILPP solvers up to 2002. Commercial MILPP solvers, such as CPLEX [10] and GUROBI [9], can be easily accessed using mathematical programming languages [3, 7, 11, 14].

The two methods to solve MILPP that are outlined in this chapter are generic, in the sense that they can be applied to any generic MILPP. For very difficult and large-scale MILPPs, decomposition techniques are often employed. These techniques exploit the structure of a problem to determine intelligent ways in which to break the MILPP into smaller subproblems from which a good solution to the overall problem can be found. Another extension of the solution methods discussed here are to develop other types of cutting planes. These cutting planes often exploit the structure of the problem at hand. While Gomory cuts are guaranteed to eventually find an optimal solution to a MILPP, the number of cuts that may need to be added grows exponentially with the problem size. Other types of cuts may yield a solution more quickly than Gomory cuts alone can. Wolsey and Nemhauser [16], Bertsimas and Weismantel [1], and Rao [13] discuss these more advanced solution techniques, including problem decomposition, cutting planes, and hybrid methods. Castillo *et al.* [4] provide further

discussion of modeling using mixed-integer linear optimization problems. We also refer interested readers to other relevant works on the topic of MILPPs [5, 12, 15].

Our discussion in this chapter focuses exclusively on mixed-integer linear optimization problems. It is a straightforward extension of the formulation techniques discussed in this chapter and in Chapter 4 to formulate mixed-integer nonlinear optimization problems. Indeed, such problems are gaining increased focused from the operations research community. However, the solution of such problems is still typically quite taxing and demanding compared to MILPPs. We refer interested readers to the work of Floudas [6], which provides an excellent introduction to the formulation and solution of mixed-integer nonlinear optimization problems.

## 3.8 GAMS Codes

This final section provides GAMS [3] codes for the main problems considered in this chapter. GAMS uses a variety of different solvers, among them CPLEX [10] and GUROBI [9], to actually solve MILPPs.

### *3.8.1 Photovoltaic Panel-Repair Problem*

The Photovoltaic Panel-Repair Problem, which is introduced in Section 3.1.1, has the following GAMS formulation:

```
1 option OPTCR=0;
2 variable z;
3 integer variables x1, x2;
4 equations of, l1, l2;
5 of .. z =e= x1+x2;
6 l1 .. 17*x1+32*x2 =l= 136;
7 l2 .. 32*x1+15*x2 =l= 120;
8 model pv /all/;
9 solve pv using mip maximizing z;
```

Line 1 indicates that the solution tolerance should be 0. Otherwise, the solver may terminate once the optimality gap is sufficiently small but non-zero, giving a near-optimal but not a fully optimal solution. Lines 2 and 3 are variable declarations, Line 4 gives names to the equations of the model, Line 5 defines the objective function, Lines 6 and 7 define the constraints, Line 8 defines the model, and Line 9 directs GAMS to solve it.

The GAMS output that provides information about the optimal solution is:

```
                           LOWER    LEVEL    UPPER   MARGINAL

---- VAR z                -INF      5.000    +INF      .
---- VAR x1                .        2.000  100.000    1.000
---- VAR x2                .        3.000  100.000    1.000
```

### 3.8.2 *Natural Gas-Storage Problem*

The Natural Gas-Storage Problem, which is introduced in Section 3.1.2, has the following GAMS formulation:

```
1  option OPTCR=0;
2  variable z;
3  positive variables y11, y12, y21, y22, y31, y32;
4  binary variables x1, x2, x3;
5  equations of, d1, d2, s1, s2, s3;
6  of .. z =e= y11+6*y12+2*y21+5*y22+3*y31+4*y32-8*x1-9*
       x2-7*x3;
7  d1 .. y11+y21+y31 =e= 10;
8  d2 .. y12+y22+y32 =e= 6;
9  s1 .. y11+y12 =l= 7*x1;
10 s2 .. y21+y22 =l= 8*x2;
11 s3 .. y31+y32 =l= 9*x3;
12 model gs /all/;
13 solve gs using mip maximizing z;
```

Line 1 indicates that the solution tolerance should be 0, Lines 2–4 are variable declarations, Line 5 gives names to the equations of the model, Line 6 defines the objective function, Lines 7–11 define the constraints, Line 12 defines the model, and Line 13 directs GAMS to solve it.

The GAMS output that provides information about the optimal solution is:

```
                           LOWER    LEVEL      UPPER     MARGINAL

---- VAR z                -INF     49.000     +INF        .
---- VAR y11               .        1.000     +INF        .
---- VAR y12               .        6.000     +INF        .
---- VAR y21               .          .       +INF        .
---- VAR y22               .          .       +INF       -2.000
---- VAR y31               .        9.000     +INF        .
---- VAR y32               .          .       +INF       -4.000
---- VAR x1                .        1.000     1.000      -8.000
---- VAR x2                .          .       1.000      -1.000
---- VAR x3                .        1.000     1.000      11.000
```

### 3.8.3 *Electricity-Scheduling Problem*

The Electricity-Scheduling Problem, which is introduced in Section 3.1.3, has the following GAMS formulation:

```
1  option OPTCR=0;
2  variable z;
3  positive variables p1, p2, p3;
4  binary variables x1, x2, x3;
5  equations of, b, l1u, l1d, l2u, l2d, l3u, l3d;
6  of  .. z =e= 2*p1+5*p2+1*p3 + 40*x1+50*x2+35*x3;
7  b   .. p1+p2+p3 =e= 50;
8  l1u .. p1 =l= 20*x1;
9  l1d .. p1 =g=  5*x1;
10 l2u .. p2 =l= 40*x2;
11 l2d .. p2 =g=  6*x2;
12 l3u .. p3 =l= 35*x3;
13 l3d .. p3 =g=  4*x3;
14 model es /all/;
15 solve es using mip minimizing z;
```

Line 1 indicates that the solution tolerance should be 0, Lines 2–4 are variable declarations, Line 5 gives names to the equations of the model, Line 6 defines the objective function, Lines 7–13 define the constraints, Line 14 defines the model, and Line 15 directs GAMS to solve it.

The GAMS output that provides information about the optimal solution is:

```
1                          LOWER    LEVEL    UPPER     MARGINAL

3  ---- VAR z              -INF   140.000    +INF         .
4  ---- VAR p1              .      15.000    +INF         .
5  ---- VAR p2              .        .       +INF       3.000
6  ---- VAR p3              .      35.000    +INF         .
7  ---- VAR x1              .       1.000    1.000     40.000
8  ---- VAR x2              .         .      1.000     50.000
9  ---- VAR x3              .       1.000    1.000       EPS
```

### 3.8.4 *Oil-Transmission Problem*

The Oil-Transmission Problem, which is introduced in Section 3.1.4, has the following GAMS formulation:

```
1  option OPTCR=0;
2  variable z;
3  positive variables p1, p2;
4  variable f1, f2, f3;
5  binary variables x1, x2;
```

```
6  equations of, b1, b2, b3, l12, l21, l13, l31, l23, l32
       ;
7  of .. z =e= 2000*p1+3000*p2-50000*x1-55000*x2;
8  b1 .. p1+p2 =e= 30;
9  b2 .. p1-f1-f3 =e=  0;
10 b3 .. p2-f2+f3 =e=  0;
11 l12 .. f3 =l=  10;
12 l21 .. f3 =g= -10;
13 l13 .. f1 =l=  12 + 11*x1;
14 l31 .. f1 =g= -12 - 11*x1;
15 l23 .. f2 =l=  11 + 12*x2;
16 l32 .. f2 =g= -11 - 12*x2;
17 model ot /all/;
18 solve ot using MIP maximizing z;
```

Line 1 indicates that the solution tolerance should be 0, Lines 2–5 are variable declarations, Line 6 gives names to the equations of the model, Line 7 defines the objective function, Lines 8–16 define the constraints, Line 17 defines the model, and Line 18 directs GAMS to solve it.

The GAMS output that provides information about the optimal solution is:

```
1                        LOWER      LEVEL    UPPER     MARGINAL

3  ---- VAR z           -INF   35000.000    +INF         .
4  ---- VAR p1           .          .       +INF   -1000.000
5  ---- VAR p2           .        30.000    +INF         .
6  ---- VAR f1          -INF      10.000    +INF         .
7  ---- VAR f2          -INF      20.000    +INF         .
8  ---- VAR f3          -INF     -10.000    +INF         .
9  ---- VAR x1           .          .       1.000  -5.000E+4
10 ---- VAR x2           .        1.000     1.000  -5.500E+4
```

### 3.8.5 *Charging-Station Problem*

The Charging-Station Problem, which is introduced in Section 3.1.5, has the following GAMS formulation:

```
1  option OPTCR=0;
2  variable z;
3  binary variables x1, x2, x3;
4  equations of, n1, n2, n3, n4;
5  of .. z =e= 10*x1+12*x2+13*x3;
6  n1 .. 1*x1 + 0*x2 + 1*x3 =g= 1;
7  n2 .. 0*x1 + 1*x2 + 0*x3 =g= 1;
8  n3 .. 1*x1 + 1*x2 + 0*x3 =g= 1;
9  n4 .. 0*x1 + 0*x2 + 1*x3 =g= 1;
10 model cs /all/;
11 solve cs using mip minimizing z;
```

Line 1 indicates that the solution tolerance should be 0, Lines 2 and 3 are variable declarations, Line 4 gives names to the equations of the model, Line 5 defines the objective function, Lines 6–9 define the constraints, Line 10 defines the model, and Line 11 directs GAMS to solve it.

The GAMS output that provides information about the optimal solution is:

```
1                          LOWER      LEVEL      UPPER      MARGINAL

3   ---- VAR z             -INF      25.000      +INF          .
4   ---- VAR x1             .           .        1.000       10.000
5   ---- VAR x2             .         1.000      1.000       12.000
6   ---- VAR x3             .         1.000      1.000       13.000
```

### 3.8.6 *Wind Farm-Maintenance Problem*

The Wind Farm-Maintenance Problem, which is introduced in Section 3.1.6, has the following GAMS formulation:

```
1  option OPTCR=0;
2  variable z;
3  binary variables x11,x12,x13,x21,x22,x23,x31,x32,x33;
4  equations of, f1, f2, f3, t1, t2, t3;
5  of .. z =e= 10*x11+12*x12+14*x13+9*x21+8*x22+15*x23
       +10*x31+5*x32+15*x33;
6  f1 .. x11 + x12 + x13 =e= 1;
7  f2 .. x21 + x22 + x23 =e= 1;
8  f3 .. x31 + x32 + x33 =e= 1;
9  t1 .. x11 + x21 + x31 =e= 1;
10 t2 .. x12 + x22 + x32 =e= 1;
11 t3 .. x13 + x23 + x33 =e= 1;
12 model wf /all/;
13 solve wf using mip minimizing z;
```

Line 1 indicates that the solution tolerance should be 0, Lines 2 and 3 are variable declarations, Line 4 gives names to the equations of the model, Line 5 defines the objective function, Lines 6–11 define the constraints, Line 12 defines the model, and Line 13 directs GAMS to solve it.

The GAMS output that provides information about the optimal solution is:

```
1                          LOWER      LEVEL      UPPER      MARGINAL

3   ---- VAR z             -INF      28.000      +INF          .
4   ---- VAR x11            .           .        1.000       10.000
5   ---- VAR x12            .           .        1.000       12.000
6   ---- VAR x13            .         1.000      1.000       14.000
7   ---- VAR x21            .         1.000      1.000        9.000
8   ---- VAR x22            .           .        1.000        8.000
9   ---- VAR x23            .           .        1.000       15.000
10  ---- VAR x31            .           .        1.000       10.000
```

```
11 ---- VAR x32             .          1.000      1.000      5.000
12 ---- VAR x33             .            .        1.000     15.000
```

## 3.9 Exercises

**3.1** Steve owns two warehouses containing 120 and 100 photovoltaic panels, respectively. He uses these two warehouses to serve customers in three markets, which have demands for 40, 70, and 50 units, respectively, which must be met exactly. Per-unit transportation costs between each warehouse and each market are given in Table 3.8. Formulate an MILPP to determine how many panels Steve should ship from each warehouse to each market to minimize total transportation cost.

**Table 3.8** Per-unit transportation costs [$] for Exercise 3.1

| | Warehouse 1 | Warehouse 2 |
|---|---|---|
| Market 1 | 14 | 12 |
| Market 2 | 13 | 10 |
| Market 3 | 11 | 11 |

**3.2** An electricity producer may use some combination of electricity-generating units that are powered by either wind or natural gas. The capacity of each unit (either wind- or natural gas-powered) is 10 kW. The cost of building a wind-powered unit is $2500 and its operating cost is $6/kWh. The cost of building a natural gas-powered unit is $2000 and its operating cost is $80/kWh.

If the demand to be supplied is 100 kW and the producer may not build more than four wind-powered units, how many units of each type should the electricity producer build to minimize the sum of building and generation costs?

**3.3** Design, formulate, and solve an assignment problem similar to the Wind Farm-Maintenance Problem, which is introduced in Section 3.1.6.

**3.4** Design, formulate, and solve a knapsack problem similar to the Photovoltaic Panel-Repair Problem, which is introduced in Section 3.1.1.

**3.5** Design, formulate, and solve a set-covering problem similar to the Charging-Station Problem, which is introduced in Section 3.1.5.

**3.6** Linearize the cost function:

$$\text{cost} = \begin{cases} 0, & \text{if } x = 0, \\ 5 + x, & \text{if } 0 < x \leq 10, \end{cases}$$

using integer variables.

**3.7** Linearize the piecewise-linear cost function:

$$\text{cost} = \begin{cases} \frac{1}{2}x, & \text{if } 0 \le x \le 1, \\ \frac{1}{2} + \frac{2}{3}(x-1), & \text{if } 1 < x \le 2, \\ \frac{7}{6} + \frac{1}{3}(x-2), & \text{if } 2 < x \le 3, \end{cases}$$

using integer variables.

**3.8** Design and formulate an instance of an Alternative Constraint, as outlined in Section 3.3.4.

**3.9** Consider the MILPP:

$$\begin{aligned} \min_{x_1,x_2} z &= -x_1 - x_2 \\ \text{s.t. } & 2x_1 + x_2 \le 13 \\ & x_1 + 2x_2 \le 12 \\ & x_1, x_2 \ge 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

The linear relaxation of this problem has the optimal solution $x^0 = (14/3, 11/3)^\top$. If the Branch-and-Bound Algorithm is to be applied where $x_1$ is the branching variable, determine the new LPPs that would be added to $\Xi$.

**3.10** Generate a cutting plane for the MILPP in Exercise 3.9.

**3.11** Solve the Photovoltaic Panel-Repair Problem using the Cutting-Plane Algorithm.

**3.12** Solve the Natural Gas-Storage Problem using the Branch-and-Bound Algorithm.

## References

1. Bertsimas D, Weismantel R (2005) Optimization over integers. Dynamic Ideas, Belmont
2. Bixby RE (2002) Solving real-world linear programs: a decade and more of progress. Oper Res 50:3–15
3. Brook A, Kendrick D, Meeraus A (1988) GAMS – a user's guide. ACM, New York. www.gams.com
4. Castillo E, Conejo AJ, Pedregal P, García R, Alguacil N (2002) Building and solving mathematical programming models in engineering and science. Pure and applied mathematics series, Wiley, New York
5. Conforti M, Cornuejols G, Zambelli G (2016) Integer programming. Springer, New York
6. Floudas CA (1995) Nonlinear and mixed-integer optimization: fundamentals and applications. Oxford University Press, Oxford

7. Fourer R, Gay DM, Kernighan BW (2002) AMPL: a modeling language for mathematical programming, 2nd edn. Cengage Learning, Boston. www.ampl.com
8. Gomory RE (1960) An algorithm for the mixed integer problem. RAND Corporation, Santa Monica
9. Gurobi Optimization, Inc. (2010) Gurobi optimizer reference manual, Version 3.0. Houston, Texas. www.gurobi.com
10. IBM ILOG CPLEX Optimization Studio (2016) CPLEX user's manual, Version 12 Release 7. IBM Corp. www.cplex.com
11. Lubin M, Dunning I (2015) Computing in operations research using Julia. INFORMS J Comput 27(2):238–248. www.juliaopt.org
12. Papadimitriou CH, Steiglitz K (1998) Combinatorial optimization: algorithms and complexity. Dover Publications, Mineola
13. Rao SS (2009) Engineering optimization: theory and practice, 4th edn. Wiley, Hoboken
14. Roelofs M, Bisschop J (2016) AIMMS the language reference, AIMMS B.V., Haarlem, The Netherlands. www.aimms.com
15. Wolsey LA (1998) Integer programming. Wiley interscience series in discrete mathematics and optimization, Wiley, New York
16. Wolsey LA, Nemhauser GL (1999) Integer and combinatorial optimization. Wiley interscience series in discrete mathematics and optimization, Wiley, New York