Considering Figure 4.1, many other formulations are also possible for the set \( S \), and readers are encouraged to draw some of them. You can see that there is a smallest formulation whose extreme points are integer. Geometrically, it is called the convex hull of the set \( S \), denoted by \( \text{Conv}(S) \). This is an ideal formulation. Special classes of combinatorial optimization problems, such as the assignment, transportation, transshipment, maximum flow, and linear minimum cost flow, have the property that their LP relaxation is the convex hull of basic feasible integer solutions. We refer to this class as “easy integer programs” to be discussed in Chapter 10. However, the ideal formulation for a general integer program is very difficult to find.

In what follows, we give two real-world examples to show how one formulation is better than the other. One example is a knapsack problem for a pure 0–1 integer program and another is an uncapacitated facility location problem for a mixed integer program.

### Example 4.1 (The Knapsack Problem)

The following two polyhedra, \( P_1 \) and \( P_2 \), are formulations for \( S \) because they satisfy \( P \subseteq E^3 \) and \( S = P \cap S \), where \( S = \{ (0, 0, 0, 0, 0, 0, 0, 0), (0, 1, 0, 0, 0), (0, 0, 1, 0, 0), (0, 0, 0, 1, 0), (0, 0, 0, 0, 1), (1, 1, 0, 0, 0), (1, 0, 1, 0, 0), (1, 0, 0, 1, 0), (0, 0, 0, 0, 1), (0, 1, 1, 0, 0), (0, 1, 0, 1, 0), (0, 1, 0, 0, 1), (0, 1, 1, 0, 1), (0, 1, 1, 0, 1), (0, 1, 1, 0, 1), (0, 1, 1, 0, 1), (0, 1, 1, 0, 1), (0, 1, 1, 0, 1), (0, 1, 1, 0, 1), \}

- \( P_1 = \{ y \in E^3 : 13y_1 + 21y_2 + 4y_3 + 17y_4 + 4y_5 \leq 47, 0 \leq y \leq 1 \} \)
- \( P_2 = \{ y \in E^3 : 3y_1 + 3y_2 + 3y_3 + 3y_4 \leq 8, 0 \leq y \leq 1 \} \)

To show that formulation \( P_2 \) is better than \( P_1 \), we must show that \( P_2 \subseteq P_1 \), which is equivalent to show that \( P_2 \subseteq P_1 \) and \( P_2 \not\subseteq P_1 \). If we can show that all the points in \( P_2 \) are also in \( P_1 \), then \( P_3 \subseteq P_1 \). In addition, if we can also show there exists a point in \( P_1 \) but not in \( P_2 \), then \( P_2 \subseteq P_1 \).

First, we show that all the points in \( P_2 \) are also in \( P_1 \). Multiplying by 4 on both sides of the constraint in \( P_2 \), we obtain an equivalent constraint

\[
12y_1 + 12y_2 + 4y_3 + 12y_4 + 4y_5 \leq 32
\]

The constraint in \( P_1 \) can be rewritten as

\[
13y_1 + 21y_2 + 4y_3 + 17y_4 + 4y_5 \leq 47
\]

or

\[
(12y_1 + 12y_2 + 4y_3 + 12y_4 + 4y_5) + (y_1 + 9y_2 + 5y_4) \leq 47
\]

If we can show that \( y_1 + 9y_2 + 5y_4 \leq 47 - 32 \), then it implies that \( P_2 \subseteq P_1 \). The claim is true because \( 0 \leq y \leq 1 \).

Next, to show that there exists a point in \( P_1 \) but not in \( P_2 \), consider the point \( y^* = (0.02, 1, 1, 1, 1) \). We have \( y^* \in P_1 \) since \( 13(0.02) + 21(1) + 4(1) + 17(1) + 4(1) = 46.26 < 47 \), but \( y^* \not\in P_2 \) since \( 3(0.02) + 3(1) + 1(1) + 3(1) + 1(1) = 8.06 > 8 \). Hence, we conclude that \( P_2 \subseteq P_1 \).
Example 4.2 (The Uncapacitated Facility Location Problem) There are \( m \) machines available to meet the production requirement from \( n \) workshops, each with a demand 1. Once a machine is set up, a fixed cost of \( f_i \) is incurred. Unit transportation cost of products from machine \( i \) to workshop \( j \) is \( c_{ij} \). The objective is to find a production plan with the lowest cost while meeting the demands at all workshops.

To model this problem, we let \( x_{ij} \) be the fraction of demand from workshop \( j \) met by machine \( i \). Also, let \( y_i \) be 1 if machine \( i \) is used, and 0 otherwise. Two alternative models are obtained.

\[
\text{(IP}_1\text{)} \quad \text{Minimize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i
\]

subject to

\[
\sum_{i=1}^{m} x_{ij} = 1 \quad j = 1, 2, \ldots, n \quad (4.1)
\]

\[
\sum_{j=1}^{n} x_{ij} \leq n y_i \quad i = 1, 2, \ldots, m \quad (4.2)
\]

\[
x_{ij} \geq 0 \quad i = 1, 2, \ldots, m; j = 1, 2, \ldots, n
\]

\[
y_i = 0 \text{ or } 1 \quad i = 1, 2, \ldots, m
\]

\[
\text{(IP}_2\text{)} \quad \text{Minimize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i
\]

subject to

\[
\sum_{i=1}^{m} x_{ij} = 1 \quad j = 1, 2, \ldots, n
\]

\[
x_{ij} \leq y_i \quad i = 1, 2, \ldots, m; j = 1, 2, \ldots, n \quad (4.3)
\]

\[
x_{ij} \geq 0 \quad i = 1, 2, \ldots, m; j = 1, 2, \ldots, n
\]

\[
y_i = 0 \text{ or } 1 \quad i = 1, 2, \ldots, m
\]

Note that the two IP models are similar except for constraint sets (4.2) and (4.3). Considering the problem size, model IP2 is larger than IP1 because the number of constraints in (4.3) is \( n \) times that of (4.2). However, we claim that formulation \( P_2 \) is better than \( P_1 \) because \( P_2 \subseteq P_1 \), where these represent the LP relaxation of the respective integer programs, hence in both models \( y_i \) becomes continuous on \([0, 1]\).

To show that \( P_2 \subseteq P_1 \), we simply need to show that any points in \( P_2 \) also lie in \( P_1 \), but not vice versa. Since the only difference in these two formulations is that (4.3) replacing (4.2) in \( P_2 \), showing \( P_2 \subseteq P_1 \) is equivalent to showing that any points satisfying (4.3) also satisfy (4.2), but not vice versa.

Clearly, if we sum the inequalities in (4.3) over the range of \( j \), then we obtain (4.2). Hence, every point satisfying (4.3) must also satisfy (4.2). On the other hand, we can easily find an example that satisfies (4.2) but not (4.3).
Consider a special case where \( m = n \). Let \( y_i = (1/n) \) for all \( i \), then \( n y_i = 1 \). The coefficient matrix of \( x \) is a diagonal matrix with all the elements on the diagonal equal to 1, and others 0:

\[
\begin{bmatrix}
1 & 0 & \ldots & 0 & 0 \\
0 & 1 & \ldots & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & \ldots & 1 & 0 \\
0 & 0 & \ldots & 0 & 1 \\
\end{bmatrix}
\]

We can see that in this case, \( \sum_j x_{ij} = 1 = ny_i \), which satisfies (4.2), but \( x_{ij} > y_i \) for each \( i \), which violates (4.3). So we can conclude that \( P_2 \subset P_1 \), and \( P_2 \) is a better formulation.

In fact, the example discussed earlier is not the only case where (4.2) is satisfied but (4.3) is violated. Readers are encouraged to make up other examples too.

### 4.2 AUTOMATIC PROBLEM PREPROCESSING

Building good formulations for a given IP problem is both art and science. It often depends on the creativity of a model builder as well as scientific techniques. Even for the same model builder, one can often expect that his/her original formulation can be improved, either artistically or scientifically. In the remaining sections, we will introduce some logical rules that can be used to automatically improve a given formulation. These rules routinely process a given problem formulation before it is actually solved by an MIP algorithm. These rules are bundled together to form the so-called preprocessor or presolver. The preprocessor has been proven very efficient in reducing the solution space and speeding up the solution time. In fact, nowadays most popular IP software has a built-in preprocessor. Most preprocessors cover the following basic functions:

1. Tightening bounds on variables
2. Fixing variables
3. Eliminating redundant constraints
4. Identifying infeasibility
5. Tightening constraints
6. Decomposing the problem into independent subproblems
7. Scaling the coefficient matrix

There are many preprocessing techniques available in the literature; the interested reader should refer to the note of this section. In Section 4.3, we introduce a basic preprocessing technique for tightening bounds, fixing variables, and identifying redundant constraints and infeasibility for general integer programs. Then in Section 4.4, we introduce basic techniques for the same functions specially designed
for pure 0–1 integer programs. Methods for decomposing the problem and scaling the coefficient matrix are given in Sections 4.5 and 4.6, respectively.

4.3 TIGHTENING BOUNDS ON VARIABLES

We introduce a basic technique based on tightening upper and lower bounds on variables in mixed integer programs. Three types of variables are considered in order: continuous, general integer, and 0–1 variables. First, we introduce a bounding technique on continuous variables as a foundation. Then we modify and simplify this bounding technique for the special treatment of general integer and 0–1 variables.

Basically, a preprocessor is initiated with the input IP model. It examines and computes possible tighter upper and lower bounds on all variables, one at a time, in the following order: constraint 1, constraint 2, . . . , constraint \( i \), . . . , constraint \( m \); and within each constraint, variable \( x_1, x_2, . . . , x_k, . . . , x_n \). If a computed upper bound of a variable is lower than the best upper bound found so far, or a computed lower bound is higher than the best lower bound found so far, then the computed bound replaces the current bound.

After an entire constraint set is evaluated, the process is terminated if there are no bound improvements on any of the variables. If any bound is improved, then a smaller or better formulation is obtained and another round (pass) of preprocessing is repeated on the new formulation. The process is repeated until no improvements are possible on either lower or upper bounds for any variables of an entire formulation. Alternatively, a termination condition may be set to a maximum number of passes predetermined by the user.

4.3.1 Bounds on Continuous Variables

The bounded linear programming problem can be stated as

Maximize \[ z = \sum_j c_j x_j \]

subject to
\[ \sum_j a_{ij} x_j \leq b_i \quad (i = 1, 2, \ldots, m) \] \hspace{1cm} (4.4)
\[ l_j \leq x_j \leq u_j \quad (j = 1, 2, \ldots, n) \]

Separating positive and negative coefficients, the constraints can be rewritten as
\[ \sum_{j:a_{ij} > 0} a_{ij} x_j + \sum_{j:a_{ij} < 0} a_{ij} x_j \leq b_i \quad (i = 1, 2, \ldots, m) \] \hspace{1cm} (4.5)

Isolating variable \( x_k \), we have
\[ a_k x_k + \sum_{j \neq k:a_{ij} > 0} a_{ij} x_j + \sum_{j \neq k:a_{ij} < 0} a_{ij} x_j \leq b_i \quad (i = 1, 2, \ldots, m) \] \hspace{1cm} (4.6)
where \( k \) is the index of the variable to be computed for possible tighter bounds, \( j \) is the index of the remaining variables, \( u_j \) and \( l_j \), respectively, denote the tightest upper and lower bounds found so far on variable \( j \). If they are not specified, we may initially let \( u_j = M \) (a big number) and \( l_j = 0 \). The upper and lower bounds on variable \( x_k \) can be computed based on (4.6).

If \( a_{ik} > 0 \), then an upper bound on \( x_k \) can be computed by

\[
\hat{u}_k = \frac{1}{a_{ik}} \left( b_i - \sum_{j \neq k, a_{ij} > 0} a_{ij} l_j - \sum_{j \neq k, a_{ij} < 0} a_{ij} u_j \right) \quad (4.7)
\]

If \( a_{ik} < 0 \), then a lower bound on \( x_k \) can be computed by

\[
\hat{l}_k = \frac{1}{a_{ik}} \left( b_i - \sum_{j \neq k, a_{ij} > 0} a_{ij} l_j - \sum_{j \neq k, a_{ij} < 0} a_{ij} u_j \right) \quad (4.8)
\]

The basic idea is that any potential tighter bound must not exclude any feasible solution even under the “worst” conditions. For a positive coefficient \( a_{ik} \) in (4.7), the worst possible conditions are \( l_j \) for positive coefficient \( a_{ij} \) and are \( u_j \) for negative \( a_{ij} \). Similarly, for a negative coefficient \( a_{ik} \) in (4.8), the possible worst conditions are \( l_j \) for positive coefficient \( a_{ij} \) and are \( u_j \) for negative \( a_{ij} \).

After calculations, the new best bounds are updated by setting \( u_k = \hat{u}_k \) if \( \hat{u}_k < u_k \) and setting \( l_k = \hat{l}_k \) if \( \hat{l}_k > l_k \).

**4.3.2 Bounds on General Integer Variables**

In the presence of integer variables, upper and lower bounds can be further tightened by rounding the fractional values. If an integer variable has a computed upper bound \( \hat{u}_k \) that is noninteger, then it can be further tightened by rounding it down to obtain the largest integer smaller than \( \hat{u}_k \), or symbolically \( x_k \leq \lfloor \hat{u}_k \rfloor \). For example, if \( u_k = 2.47 \), then 2 is a tighter upper bound for integer \( x_k \).

If an integer variable has a lower bound \( \hat{l}_k \) that is noninteger, then it can be tightened by rounding it up to obtain the smallest integer greater than \( \hat{l}_k \), or symbolically \( x_k \geq \lceil \hat{l}_k \rceil \). For example, if \( \hat{l}_k = 0.3 \), then 1 is a lower bound for integer \( x_k \). The current best bound will then be replaced (updated) if the computed bound is tighter.

**Example 4.3 (MIP Problem)**

\[
\begin{align*}
4x_1 - 3x_2 - 2x_3 + y_4 + 2y_5 &\leq 13 \\
-3x_1 + 2x_2 - x_3 + 2y_4 + 3y_5 &\leq -9 \\
x_1 &\geq 0 \\
0 &\leq x_2 \leq 3 \\
1 &\leq x_3 \leq 5 \\
2 &\leq y_4 \leq 4 \\
y_5 &\geq 0 \text{ and integer}
\end{align*}
\]
TIGHTENING BOUNDS ON VARIABLES

Initialization:

\[ u_1 = M, \ l_1 = 0, \ u_2 = 3, \ l_2 = 0, \ u_3 = 5, \ l_3 = 1, \ u_4 = 4, \ l_4 = 2, \ u_5 = M, \ l_5 = 0 \]

Iteration 1

Check constraint 1:

\[ x_1 : u_1 = (13-1(2)-2(0)+3(3)+2(5))/4 = 7.5 < M, \text{ so } u_1 \text{ is updated to } 7.5 \]
\[ x_2 : l_2 = (13-4(0)-1(2)-2(0)+2(5))/(-3) = -7 < 0, \text{ so } l_2 \text{ is not updated} \]
\[ x_3 : l_3 = (13-4(0)-1(2)-2(0)+3(3))/(-2) = -10 < 1, \text{ so } l_3 \text{ is not updated} \]
\[ y_4 : u_4 = [(13-4(0)-2(0)+3(3)+2(5))/1] = 32 > 4, \text{ so } u_4 \text{ is not updated} \]
\[ y_5 : u_5 = [(13-4(0)-2(0)+3(3)+1(4))/2] = 13 < M, \text{ so } u_5 \text{ is updated to } 13 \]

Check constraint 2:

\[ x_1 : l_1 = (-9-2(0)-2(2)-3(0)+1(5))/(-3) = 2.67 > 0, \text{ so } l_1 \text{ is updated to } 2.67 \]
\[ x_2 : u_2 = (-9-2(2)-3(0)+3(7.5)+1(5))/2 = 7.25 > 3, \text{ so } u_2 \text{ is not updated} \]
\[ x_3 : l_3 = (-9-2(0)-2(2)-3(0)+3(7.5))/(-3) = -3.17 < 1, \text{ so } l_3 \text{ is not updated} \]
\[ y_4 : u_4 = [(-9-2(0)-3(0)+3(7.5)+1(5))/2] = 9 > 4, \text{ so } u_4 \text{ is not updated} \]
\[ y_5 : u_5 = [(-9-2(0)-2(2)+3(7.5)+1(5))/3] = 4 < 13, \text{ so } u_5 \text{ is updated to } 4 \]

After the first iteration, we have

\[ 2.67 \leq x_1 \leq 7.5, \quad 0 \leq x_2 \leq 3, \quad 1 \leq x_3 \leq 5, \quad 2 \leq y_4 \leq 4, \quad 0 \leq y_5 \leq 4 \]

Iteration 2

Check constraint 1:

\[ x_1 : u_1 = (13-1(2)-2(0)+3(3)+2(5))/4 = 7.5, \text{ so } u_1 \text{ is not updated} \]
\[ x_2 : l_2 = (13-4(2.67)-1(2)-2(0)+2(5))/(-3) = -3.44 < 0, \text{ so } l_2 \text{ is not updated} \]
\[ x_3 : l_3 = (13-4(2.67)-1(2)-2(0)+3(3))/(-2) = -3.11 < 1.67, \text{ so } l_3 \text{ is not updated} \]
\[ y_4 : u_4 = [(13-4(2.67)-2(0)+3(3)+2(5))/1] = 21 > 4, \text{ so } u_4 \text{ is not updated} \]
\[ y_5 : u_5 = [(13-4(2.67)-2(0)+3(3)+1(4))/2] = 7 > 4, \text{ so } u_5 \text{ is not updated} \]

Check constraint 2:

\[ x_1 : l_1 = (-9-2(0)-2(2)-3(0)+1(5))/(-3) = 2.67, \text{ so } l_1 \text{ is not updated} \]
\[ x_2 : u_2 = (-9-2(2)-3(0)+3(7.5)+1(5))/2 = 7.25 > 3, \text{ so } u_2 \text{ is not updated} \]
\[ x_3 : l_3 = (-9-2(0)-2(2)-3(0)+3(7.5))/(-3) = -3.17 < 1, \text{ so } l_3 \text{ is not updated} \]
\[ y_4 : u_4 = [(-9-2(0)-3(0)+3(7.5)+1(5))/2] = 9 > 4, \text{ so } u_4 \text{ is not updated} \]
\[ y_5 : u_5 = [(-9-2(0)-2(2)+3(7.5)+1(5))/3] = 4, \text{ so } u_5 \text{ is not updated} \]

Stop, because no bounds that can be further tightened were found in this iteration.
4.3.3 Bounds on 0–1 Variables

Recall that for a 0–1 variable, the worst lower bound is 0 and the worst upper bound is 1. Thus, initially we can set \( l_j = 0 \) and \( u_j = 1 \) for all \( j \). Any fractional upper bound may be rounded down to 0 and any fractional lower bound may be rounded up to 1. By rounding, for example, if \( u_1 = 0.37 \), then 0 is a new upper bound for binary variable \( y_1 \); and if \( l_2 = 0.42 \), then 1 is a new lower bound for binary variable \( y_2 \).

**Example 4.4 (BIP Problem)**

\[
\begin{align*}
8y_1 + 11y_2 - 9y_3 + 4y_4 & \leq 0 \\
y_1 - 4y_2 - 6y_3 + y_4 & \leq -5 \\
\text{all } y_j & = 0 \text{ or } 1
\end{align*}
\]

*Iteration 1*

Check constraint 1:

- \( u_1 = \lfloor (0 - 11(0) - 4(0) + 9(1))/8 \rfloor = 1 \), so \( u_1 \) is not updated
- \( u_2 = \lfloor (0 - 8(0) - 4(0) + 9(1))/11 \rfloor = 0 \), so \( y_2 \) is fixed to 0

Substituting \( y_2 = 0 \) to the given problem, we have a new constraint set

\[
\begin{align*}
8y_1 - 9y_3 + 4y_4 & \leq 0 \\
y_1 - 6y_3 + y_4 & \leq -5
\end{align*}
\]

*Iteration 2*

Check constraint 1:

- \( u_1 = \lfloor (0 - 4(0) + 9(1))/8 \rfloor = 1 \), so \( u_1 \) is not updated
- \( l_3 = \lceil (0 - 8(0) - 4(0))/(-9) \rceil = 0 \), so \( l_3 \) is not updated
- \( u_4 = \lfloor (0 - 8(0) + 9(1))/4 \rfloor = 2 > 1 \), so \( u_4 \) is not updated

Check constraint 2:

- \( u_1 = \lfloor (-5 - 1(0) + 6(1))/1 \rfloor = 1 \), so \( u_1 \) is not updated
- \( l_3 = \lceil (-5 - 1(0) - 1(0))/(-6) \rceil = 1 \), so \( y_3 \) is fixed at 1

Substituting \( y_3 = 1 \), we have

\[
\begin{align*}
8y_1 + 4y_4 & \leq 9 \\
y_1 + y_4 & \leq 1
\end{align*}
\]
Iteration 3

Check constraint 1:

\[ u_1 = \left\lfloor \frac{(9-4(0))/9} \right\rfloor = 1, \text{ so } u_1 \text{ is not updated} \]
\[ u_4 = \left\lfloor \frac{(9-8(0))/4} \right\rfloor = 2 > 1, \text{ so } u_4 \text{ is not updated} \]

Check constraint 2:

\[ u_1 = \left\lfloor \frac{(1-1(0))/1} \right\rfloor = 1, \text{ so } u_1 \text{ is not updated} \]
\[ u_4 = \left\lfloor \frac{(1-1(0))/1} \right\rfloor = 1, \text{ so } u_4 \text{ is not updated} \]

Stop, because no tighter bounds can be obtained in both constraints. The final set of constraints remains the same as start of Iteration 3.

\[ 8y_1 + 4y_4 \leq 9 \]
\[ y_1 + y_4 \leq 1 \]
\[ y_2 = 0 \]
\[ y_3 = 1 \]

4.3.4 Variables Fixing, Redundant Constraints, and Infeasibility

There are a number of variable fixing techniques available. We discuss three of them. First, for a maximization problem in the form given in (4.4), if \( a_{ij} > 0 \) for all \( i = 1, 2, \ldots, m \) and \( c_j < 0 \), then fix \( x_j \) at \( l_j \). If \( a_{ij} < 0 \) for all \( i = 1, 2, \ldots, m \) and \( c_j > 0 \), then fix \( x_j \) at \( u_j \). Second, if the best bounds on any variable obtained after applying the bound tightening routine having \( l_k^* = u_k^* \), then variable \( x_k \) can be fixed at \( l_k^* \). Third, based on bounds on the left-hand side of a constraint, we can fix variables under the condition described below. Once a variable is fixed, it can be removed by substituting its fixed value into the current formulation (model), resulting in a smaller feasible region.

For the \( i \)th constraint, define the following upper and lower row bounds:

\[ U_i = \sum_{j: a_{ij} > 0} a_{ij}u_j + \sum_{j: a_{ij} < 0} a_{ij}l_j \]
\[ L_i = \sum_{j: a_{ij} > 0} a_{ij}l_j + \sum_{j: a_{ij} < 0} a_{ij}u_j \]

Note that \( U_i \) is an upper bound for the left-hand-side of the \( i \)th constraint (or row) and \( L_i \) is a lower bound for the left-hand-side of the \( i \)th constraint (or row). Comparing with the right-hand side \( b_i \), these row bounds can be used to (a) identify a redundant constraint, (b) identify an infeasible constraint, and (c) fix variables. Normally, we have

\[ L_i \leq b_i \leq U_i \]
Consider the following three cases outside the above bounds:

(a) If \( b_i \geq U_i \), then the \( i \)th constraint is redundant and can be removed from the problem.

(b) If \( b_i < L_i \), then the \( i \)th constraint cannot be satisfied and no feasible solution exists.

(c) If \( b_i = L_i \), then all \( x_j \) with \( a_{ij} > 0 \) can be fixed at \( x_j = l_j \), and all \( x_j \) with \( a_{ij} < 0 \) can be fixed at \( x_j = u_j \).

Example 4.5  Consider the following constraint set and bounds on variables

\[
\begin{align*}
x_1 + x_2 + x_3 - 2x_4 & \leq -6 \\
-x_1 - 3x_2 + 2x_3 - x_4 & \leq 4 \\
-x_1 + x_2 + x_4 & \leq 0 \\
0 & \leq x_1 \leq 2 \\
0 & \leq x_2 \leq 1 \\
1 & \leq x_3 \leq 2 \\
2 & \leq x_4 \leq 3
\end{align*}
\]

Compute \( U_1 = 1(2) + 1(1) + 1(2) - 2(2) = 1 \)

\( L_1 = 1(0) + 1(0) + 1(1) - 2(3) = -5 \)

Constraint 1 is infeasible since \( b_1 < L_1 \).

Example 4.6  Consider the following constraint set and bounds on variables:

\[
\begin{align*}
x_1 + x_2 + x_3 - 2x_4 & \leq -1 \\
-x_1 - 3x_2 + 2x_3 - x_4 & \leq 4 \\
-x_1 + x_2 + x_4 & \leq 0 \\
0 & \leq x_1 \leq 2 \\
0 & \leq x_2 \leq 1 \\
1 & \leq x_3 \leq 2 \\
2 & \leq x_4 \leq 3
\end{align*}
\]

Compute \( U_1 = 2 + 1 + 2 - 2(2) = 1 \)

\( L_1 = 0 + 0 + 1 - 2(3) = -5 \)

No action is taken since \(-5 \leq -1 \leq 1\).

Compute \( U_2 = -0 - 3(0) + 2(2) - 2 = 2 \)

\( L_2 = -2 - 3(1) + 2(1) - 3 = -6 \)

Thus, constraint 2 is redundant since \( b_2 > U_2 \). Remove constraint 2 and continue.

Compute \( U_3 = -0 + 1 + 3 = 4 \)

\( L_3 = -2 + 0 + 2 = 0 \)
Since $b_3 = L_3$, we can fix variables: $x_1 = u_1 = 2$, $x_2 = l_2 = 0$, and $x_4 = l_4 = 2$. Substituting these fixed values, constraint 3 reduces to

$$2 + 0 + x_3 - 2(2) \leq -1$$

or

$$x_3 \leq 1$$

Combined with the given bound $x_3 \geq 1$, we have $x_3 = 1$. Since all variables are fixed, the problem is solved.

To illustrate the first-mentioned variable fixing technique, let us assume $x_3 \leq 2$ instead of $x_3 \leq 1$, which leads to $1 \leq x_3 \leq 2$. Then we can determine the value of $x_3$ using the given objective function. For the maximization problem, if the associated $c_3 > 0$, then $x_3 = u_3 = 2$. If $c_3 < 0$, then $x_3 = l_3 = 1$.

Although the row bounding technique can also be applied even before the variable bounding routine, the power of this technique depends on the tightness of bounds on variables.

### 4.4 PREPROCESSING PURE 0–1 INTEGER PROGRAMS

Problem preprocessing is most effective when a given model is a pure 0–1 integer program, which arises frequently in combinatorial optimization problems (see Chapters 5 and 6). Problem preprocessing includes the following functions for pure 0–1 integer programs:

- Fixing 0–1 variables
- Detecting redundant constraints and infeasibility
- Tightening constraints (coefficients reduction)
- Generating cutting planes (from minimum cover)
- Rounding by division with GCD

For distinction within an MIP problem, in this section we shall use $y_j$ instead of $x_j$ to denote a 0–1 variable in a pure 0–1 integer program.

#### 4.4.1 Fixing 0–1 Variables

Isolating a variable $y_k$ and separating positive and negative coefficients of the other variables, we can rewrite the standard form of the constraint set as

$$a_k y_k + \sum_{j \neq k, a_j > 0} a_j y_j + \sum_{j \neq k, a_j < 0} a_j y_j \leq b_i \quad (i = 1, 2, \ldots, m)$$

(4.9)

Any constraint of $\geq$ form can be converted to a corresponding constraint of $\leq$ form by multiplying by $(-1)$. Note that the right-hand side constant may be negative. For fixing a 0–1 variable, the following two rules are applied to each constraint $i$: 

1. \textbf{Rule 1:} If $a_k y_k \geq 0$, then $y_k = 1$.
2. \textbf{Rule 2:} If $a_k y_k \leq 0$, then $y_k = 0$. 

These rules are applied to each constraint $i$ to determine the value of $y_k$. 

By applying these rules, we can fix the variable $y_k$ and simplify the constraint set.
Rule 1: Identify the variable (say \( y_k \)) with the largest positive coefficient (say \( a_{ik} > 0 \)).
If the sum of \( a_{ik} \) and all \( a_{ij} < 0 \) exceeds \( b_i \), then constraint \( i \) is violated at \( y_k = 1 \) and hence \( y_k \) should be fixed at 0.

Rule 2: Identify the variable (say \( y_k \)) with the most negative coefficient (say \( a_{ik} < 0 \)).
If the sum of all \( a_{ij} < 0 (j \neq k) \) exceeds \( b_i \), then constraint \( i \) is violated at \( y_k = 0 \) and hence \( y_k \) should be fixed at 1.

Note that in rule 1, the sum of \( a_{ik} \) and all \( a_{ij} < 0 \) is equivalent to setting \( y_k = 1, y_j = 1 \) if its coefficient \( a_{ij} < 0 \), and \( y_j = 0 \) if \( a_{ij} > 0 \) for all \( j \neq k \) in (4.9). In rule 2, the sum of all \( a_{ij} < 0 \) is equivalent to setting \( y_k = 0, y_j = 1 \) if \( a_{ij} < 0 \), and \( y_j = 0 \) if \( a_{ij} > 0 \) for all \( j \neq k \) in (4.9).

Consider the following example in \( \leq \) form:

\[
6y_1 + 2y_2 - 2y_3 - y_4 \leq 2
\]

Identify \( y_1 \) as the variable having the largest positive coefficient and apply rule 1. Since \( 6 + (-2) + (-1) = 3 > 2 \) violates the constraint, \( y_1 \) must be fixed at 0. Identify \( y_3 \) as the variable having the most negative coefficient. Rule 2 cannot be applied because \( -1 \leq 2 \).

Consider the following constraint in \( \geq \) form:

\[
3y_1 + y_2 - 3y_3 \geq 2
\]

Multiplying \( (-1) \) through the constraint, we obtain

\[
-3y_1 - y_2 + 3y_3 \leq -2
\]

Identify \( y_3 \) as the variable with the largest positive coefficient and apply rule 1. Since \( 3 + (-3) + (-1) > -2 \) violates the constraint, \( y_3 \) must be fixed at 0. Identify \( y_1 \) as the variable with the most negative coefficient and apply rule 2. Since \( -1 > -\infty \), \( y_1 \) must be fixed at 0.

Once a variable is fixed at 0 or 1 using a certain constraint, the fixed value can be substituted into the other constraints, which results in problem reduction.

Example 4.4 (Continued)

\[
8y_1 + 11y_2 - 9y_3 + 4y_4 \leq 0
\]
\[
y_1 - 4y_2 - 6y_3 + y_4 \leq -5
\]
all \( y_j = 0 \) or \( 1 \)

By applying rules 1 and 2 to constraint 1, we can fix \( y_2 = 0 \) and \( y_3 = 1 \) resulting in the formulation

\[
8y_1 + 4y_4 \leq 9
\]
\[
y_1 + y_4 \leq 1
\]
\[
y_2 = 0
\]
\[
y_3 = 1
\]
which, coincidently, is the same set of reduced constraints as obtained by the bound tightening technique described in the previous section.

Moreover, fixing a variable from one constraint can sometimes generate a \textit{chain reaction} of fixing other variables from other constraints. Example 4.7 presents an extension of Example 4.4 in which two constraints and three variables are added.

\textbf{Example 4.7} \quad The set of constraints include two constraints in Example 4.4 plus the following constraints:

\[
\begin{align*}
y_3 + y_4 + y_5 & \leq 1 \\
y_5 - y_6 & \geq 0
\end{align*}
\]

Variable fixing in Example 4.4 yields the reduced constraints

\[
\begin{align*}
8y_1 + 4y_4 & \leq 9 \\
y_1 + y_4 & \leq 1
\end{align*}
\]

by fixing \( y_2 = 0 \) and \( y_3 = 1 \). Next, continue the fixing process for two additional constraints.

\[
y_3 + y_4 + y_5 \leq 1 \text{ implies } y_4 = y_5 = 0, \text{ and } y_5 - y_6 \geq 0 \text{ implies } y_6 = 0
\]

Fixing variables can achieve a drastic reduction on the size of a pure 0–1 integer program. Crowder et al. (1983) reported that a problem of 2756 variables has been reduced to a problem of 1415 variables.

\subsection*{4.4.2 Detecting Redundant Constraints and Infeasibility}

There are many techniques that can be used to detect a redundant constraint. The technique presented in the previous section is based on row bounding. Here we present another one that is based on a similar idea for variable fixing as presented in rules 1 and 2 in Section 4.4.1.

\textit{Rule 3:} For a \( \leq \) constraint, assign a value of 1 to the variables with positive coefficients and 0 otherwise. If the constraint is still satisfied, then it is redundant and can be dropped from further consideration.

Again, if a constraint is in \( \geq \) form, convert it to one in \( \leq \) form by multiplying by \((-1)\). For example,

\[
2x_1 + x_2 + 3x_3 \leq 7
\]

is redundant, since \( 2(2) + 1(1) + 3(1) = 6 < 7 \). As another example,

\[
3x_1 - 2x_2 - x_3 \leq 0
\]

is redundant, since \( 3(1) - 2(0) - 1(0) = 3 \leq 3 \).
Very often redundant constraints are not detected from the original model, but are detected from the reduced models after fixing some variables. Note that the two techniques presented in this and the last sections, as well as other techniques, do not ensure detecting all redundant constraints. There are many more techniques in the literature for detecting redundant constraints.

4.4.3 Tightening Constraints (or Coefficients Reduction)

We use a flowchart in Figure 4.5 to demonstrate a constraint tightening procedure. Suppose we are given a constraint of the form

\[ a_1y_1 + a_2y_2 + \cdots + a_ny_n \leq b \quad \text{where } y_j = 0 \text{ or } 1 \text{ for all } j \]

Example 4.8 Tighten the following constraint

\[ 6y_1 + 3y_2 - 5y_3 + 2y_4 + 7y_5 - 4y_6 \leq 15 \]

Iteration 1

Calculate \( M = 6 + 3 + 2 + 7 = 18, \ M - b = 18 - 15 = 3 \).

\( S = \{ a_1, a_3, a_5, a_6 \} \).

Let \( M = \sum_{a_i > 0} a_i \). 

\[ S = \{ a_i : |a_i| > M - b \} \]

\[ S = \emptyset ? \]

\( Y \) Stop, constraint cannot be tightened

\( N \)

Select \( a_k \) from \( S \)

\[ a_k > 0 ? \]

\( Y \)

Let \( \hat{a}_k = M - b, \hat{b} = M - a_k \)

Replace \( a_k \) with \( \hat{a}_k \), \( b \) with \( \hat{b} \)

\( N \)

Let \( \hat{a}_k = b - M \)

Replace \( a_k \) with \( \hat{a}_k \)

FIGURE 4.5 Process for coefficient reduction.
PREPROCESSING PURE 0–1 INTEGER PROGRAMS

Pick $a_1$ to begin with. Since $a_1 > 0$, calculate $\hat{a}_1 = M - b = 3$, $\hat{b} = M - a_1 = 12$.
Thus, the given constraint is tightened as

$$3y_1 + 3y_2 - 5y_3 + 2y_4 + 7y_5 - 4y_6 \leq 12$$

**Iteration 2**

$M = 3 + 3 + 2 + 7 = 15$. $M - b = 15 - 12 = 3$.
$S = \{a_3, a_5, a_6\}$.
Pick $a_3$ to start with. Since $a_3 < 0$, calculate $\hat{a}_3 = b - M = -3$.
The tightened constraint becomes

$$3y_1 + 3y_2 - 3y_3 + 2y_4 + 7y_5 - 4y_6 \leq 12$$

**Iteration 3**

$M = 3 + 3 + 2 + 7 = 15$. $M - b = 15 - 12 = 3$.
$S = \{a_5, a_6\}$.
Pick $a_5$ to start with. Since $a_5 > 0$, calculate $\hat{a}_5 = M - b = 3$, $\hat{b} = M - a_5 = 8$.
The tightened constraint becomes

$$3y_1 + 3y_2 - 3y_3 + 2y_4 + 3y_5 - 4y_6 \leq 8$$

**Iteration 4**

$M = 3 + 3 + 2 + 3 = 11$. $M - b = 11 - 8 = 3$.
$S = \{a_6\}$.
Since $a_6 < 0$, calculate $\hat{a}_6 = b - M = -3$.
The tightened constraint becomes

$$y_1 + 3y_2 - 3y_3 + 2y_4 + 3y_5 - 3y_6 \leq 8$$

**Iteration 5**

$S = \emptyset$. Stop, the constraint cannot be further tightened.

### 4.4.4 Generating Cutting Planes from Minimum Cover

A cutting plane (or cut) for an IP problem is a derived constraint that reduces the feasible region for the LP relaxation without eliminating any feasible solution for the IP problem. Here we will see a particular type of cutting planes for pure 0–1 integer programs. Such a cut is generated from a constraint in $\leq$ form with all coefficients and the right-hand side positive,

$$a_1y_1 + a_2y_2 + \cdots + a_ny_n \leq b$$