Model Building in Mathematical Programming

Third Edition

H. P. WILLIAMS
Faculty of Mathematical Studies
University of Southampton

A Wiley–Interscience Publication

JOHN WILEY & SONS
Chichester · New York · Brisbane · Toronto · Singapore
CHAPTER 9

Building Integer Programming Models I

9.1 The Uses of Discrete Variables

When integer variables are used in a mathematical programming model they may serve a number of purposes. These are described below.

Indivisible (Discrete) Quantities

This is the obvious use mentioned at the beginning of Chapter 8 where we wish to use a variable to represent a quantity which can only come in whole numbers such as aeroplanes, cars, houses or men.

Decision Variables

Variables are frequently used in integer programming (IP) to indicate which of a number of possible decisions should be made. Usually these variables can only take the two values, zero or one. Such variables are known as zero–one (0–1) variables. For example, $\delta = 1$ indicates that a depot should be built and $\delta = 0$ indicates that a depot should not be built. We will usually adopt the convention of using the Greek letter ‘$\delta$’ for 0–1 variables and reserving Latin letters for continuous (real or rational) variables.

It is easy to ensure that a variable, which is also specified to be integer, can only take the two values 0 or 1 by giving the variable a simple upper bound (SUB) of 1. (All variables are assumed to have a simple lower bound of 0 unless it is stated to the contrary).

Although decision variables are usually 0–1, they need not always be. For example, we might have $\gamma = 0$ indicates that no depot should be built; $\gamma = 1$ indicates that a depot of type A should be built; $\gamma = 2$ indicates that a depot of type B should be built.

Indicator Variables

When extra conditions are imposed on a linear programming (LP) model, 0–1 variables are usually introduced and ‘linked’ to some of the continuous variables in the problem to indicate certain states. For example, suppose that $x$ represents the quantity of an ingredient to be included in a blend. We may well wish to use an indicator variable $\delta$ to distinguish between the state where $x = 0$ and the state where $x > 0$. By introducing the following constraint we can force $\delta$ to take the value 1 when $x > 0$:

$$x - M\delta \leq 0. \quad (1)$$

$M$ is a constant coefficient representing a known upper bound for $x$.

Logically we have achieved the condition

$$x > 0 \rightarrow \delta = 1, \quad (2)$$

where ‘$\rightarrow$’ stands for ‘implies’.

In many applications (2) provides a sufficient link between $x$ and $\delta$ (e.g. Example 1 below). There are applications (e.g. Example 2 below), however, where we also wish to impose the condition

$$x = 0 \rightarrow \delta = 0. \quad (3)$$

(3) is another way of saying

$$\delta = 1 \rightarrow x > 0. \quad (4)$$

Together (2) and (3) (or (4)) impose the condition

$$\delta = 1 \leftrightarrow x > 0, \quad (5)$$

where ‘$\leftrightarrow$’ stands for ‘if and only if’.

It is not possible totally to represent (3) (or (4)) by a constraint. On reflection this is not surprising. (4) gives the condition ‘if $\delta = 1$ the ingredient represented by $x$ must appear in the blend’. Would we really want to distinguish in practice between no usage of the ingredient and, say, one molecule of the ingredient? It would be much more realistic to define some threshold level $m$ below which we will regard the ingredient as unused. (4) can now be rewritten as

$$\delta = 1 \rightarrow x > m. \quad (6)$$

This condition can be imposed by the constraint

$$x - m\delta \geq 0. \quad (7)$$

Example 1. The Fixed Charge Problem

$x$ represents the quantity of a product to be manufactured at a marginal cost per unit of $C_1$. In addition, if the product is manufactured at all there is a setup cost of $C_2$. The position is summarized as follows:

$$\begin{align*}
x = 0, & \quad \text{total cost} = 0; \\
x > 0, & \quad \text{total cost} = C_1 x + C_2.
\end{align*}$$

The situation can be represented graphically as in Figure 9.1.

Clearly the total cost is not a linear function of $x$. It is not even a continuous
function as there is a discontinuity at the origin. Conventional LP is not capable of handling this situation.

In order to use IP we introduce an indicator variable \( \delta \) so that if any of the product is manufactured \( \delta = 1 \). This can be achieved by constraint (1) above. The variable \( \delta \) is given a cost of \( C_2 \) in the objective function giving the following expression for the total cost:

\[
\text{total cost} = C_1 x + C_2 \delta.
\]

By the introduction of 0–1 variables such as \( \delta \) and extra constraints such as (1), to link these variables to the continuous variables such as \( x \), fixed charges can be introduced into a model if the \( \delta \) variables are given objective coefficients equal to the fixed charges.

It is worth pointing out that this is a situation where it is not generally necessary to model the condition (3). This condition will automatically be satisfied at optimality if the objective of the model has the effect of minimizing cost. Although a solution \( x = 0, \delta = 1 \) does not violate the constraints it is clearly non-optimal since \( x = 0, \delta = 0 \) will not violate the constraints either but will result in a smaller total cost.

It would certainly not be invalid to impose condition (3) explicitly by a constraint such as (6) (as long as \( m \) was sufficiently small). In certain circumstances it might even be computationally desirable.

**Example 2. Blending**

(This example is relevant to the FOOD MANUFACTURE 2 problem in Part 2.)

\( x_a \) represents the proportion of ingredient A to be included in a blend; \( x_B \) represents the proportion of ingredient B to be included in a blend.

In addition to the conventional quality constraints (for which LP can be used) connecting these and other variables in the model it is wished to impose the following extra condition: *if A is included in the blend B must be included also.*

IP must be used to model this extra condition. A 0–1 indicator variable \( \delta \) is introduced which will take the value 1 if \( x_a > 0 \). This is linked to variable \( x_a \) by the following constraint of type (1):

\[
x_a - \delta \leq 0. \tag{8}
\]

Here the coefficient \( M \) of constraint (1) can conveniently be taken as 1 since we are dealing with proportions.

We are now in a position to use the new 0–1 variable \( \delta \) to impose the condition

\[
\delta = 1 \rightarrow x_B > 0. \tag{9}
\]

In order to impose this condition of (4) we must choose some proportionate level \( m \) (say 1/100) below which we will regard B as out of the blend. This gives us the following constraint:

\[
x_B - 0.01 \delta \geq 0. \tag{10}
\]

We have now imposed the extra condition on the LP model by introducing a 0–1 variable \( \delta \) with two extra constraints (8) and (10).

Notice that here (unlike Example 1) it was necessary to introduce a constraint to represent a condition of type (4). The satisfaction of such a condition could not be guaranteed by optimality. An extension of the extra condition which we have imposed might be the following: *if A is included in the blend B must be included also and vice versa.* This requires two extra constraints which the reader might like to formulate.

It should be pointed out that any constant coefficient \( M \) can be chosen in constraints of type (1) so long as \( M \) is sufficiently big not to restrict the value of \( x \) to an extent not desired in the problem being modelled. In practical situations it is usually possible to specify such a value for \( M \). Although theoretically any sufficiently large value of \( M \) will suffice there is computational advantage in making \( M \) as realistic as possible. This point is explained further in Section 10.1 of Chapter 10. Similar considerations apply to the coefficient \( m \) in constraints of type (7).

It is possible to use indicator variables in a similar way to show whether an inequality holds or does not hold. First, suppose that we wish to indicate whether the following inequality holds by means of an indicator variable \( \delta \):

\[
\sum_j a_j x_j \leq b.
\]

The following condition is fairly straightforward to formulate. We will
therefore model it first:
\[ \delta = 1 \rightarrow \sum_j a_j x_j \leq b. \]  \hspace{1cm} (11)

(11) can be represented by the constraint
\[ \sum_j a_j x_j + M \delta \leq M + b, \]  \hspace{1cm} (12)

where \( M \) is an upper bound for the expression \( \sum_j a_j x_j - b \). It is easy to verify that (12) has the desired effect, i.e. when \( \delta = 1 \) the original constraint is forced to hold and when \( \delta = 0 \) no constraint is implied.

A convenient way of constructing (12) from condition (11) is to pursue the following train of reasoning. If \( \delta = 1 \) we wish to have \( \sum_j a_j x_j - b \leq 0 \), i.e. if \( (1 - \delta) = 0 \) we wish to have \( \sum_j a_j x_j - b \leq 0 \). This condition is imposed if
\[ \sum_j a_j x_j - b \leq M(1 - \delta), \]

where \( M \) is a sufficiently large number. In order to find how large \( M \) must be we consider the case \( \delta = 0 \) giving \( \sum_j a_j x_j - b \leq M \).

This shows that we must choose \( M \) sufficiently large that this does not give an undesired constraint. Clearly \( M \) must be chosen to be an upper bound for the expression \( \sum_j a_j x_j - b \). Re-arranging the constraint we have obtained with the variables on the left we obtain (12).

We will now consider how to model the reverse of constraint (11), i.e.
\[ \sum_j a_j x_j \leq b \rightarrow \delta = 1. \]  \hspace{1cm} (13)

This is conveniently expressed as
\[ \delta = 0 \rightarrow \sum_j a_j x_j \not\leq b, \]  \hspace{1cm} (14)

i.e.
\[ \delta = 0 \rightarrow \sum_j a_j x_j > b. \]  \hspace{1cm} (15)

In dealing with the expression \( \sum_j a_j x_j > b \) we run into the same difficulties that we met with the expression \( x > 0 \). We must rewrite
\[ \sum_j a_j x_j > b \ as \sum_j a_j x_j \geq b + \epsilon, \]

where \( \epsilon \) is some small tolerance value beyond which we will regard the constraint as having been broken. Should the coefficients \( a_i \) be integers as well as the variables \( x_i \), as often happens in this type of situation, there is no difficulty as \( \epsilon \) can be taken as 1.

(15) may now be rewritten as
\[ \delta = 0 \rightarrow -\sum_j a_j x_j + b + \epsilon \leq 0. \]  \hspace{1cm} (16)

Using an argument similar to that above we can represent this condition by the constraint
\[ \sum_j a_j x_j - (m - \epsilon) \delta \geq b + \epsilon, \]  \hspace{1cm} (17)

where \( m \) is a lower bound for expression
\[ \sum_j a_j x_j - b. \]

Should we wish to indicate whether a \( \geq \) inequality such as
\[ \sum_j a_j x_j \geq b \]
holds or not by means of an indicator variable \( \delta \), the required constraints can easily be obtained by transforming the above constraint into a \( \leq \) form. The corresponding constraints to (12) and (17) above are
\[ \sum_j a_j x_j + m \delta \geq m + b, \]  \hspace{1cm} (18)

\[ \sum_j a_j x_j - (M + \epsilon) \delta \leq b - \epsilon, \]  \hspace{1cm} (19)

where \( m \) and \( M \) are again lower and upper bounds respectively on the expression
\[ \sum_j a_j x_j - b. \]

Finally, to use an indicator variable \( \delta \) for an \( = \) constraint such as
\[ \sum_j a_j x_j = b \]
is slightly more complicated. We can use \( \delta = 1 \) to indicate that the \( \leq \) and \( \geq \) cases hold simultaneously. This is done by stating both (12) and (18) together.

If \( \delta = 0 \) we want to force either the \( \leq \) or the \( \geq \) constraint to be broken. This may be done by expressing (17) and (19) with two variables \( \delta' \) and \( \delta'' \) giving
\[ \sum_j a_j x_j - (m - \epsilon) \delta' \geq b + \epsilon, \]  \hspace{1cm} (20)

\[ \sum_j a_j x_j - (M + \epsilon) \delta'' \leq b - \epsilon. \]  \hspace{1cm} (21)
The indicator variable \( \delta \) forces the required condition by the extra constraint
\[
\delta^* + \delta^* - \delta \leq 1. \tag{22}
\]
In some circumstances we wish to impose a condition of type (11). Alternatively we may wish to impose a condition of type (13) or impose both conditions together. These conditions can be dealt with by the linear constraints (12) or (17) taken individually or together.

**Example 3**

Use a 0–1 variable \( \delta \) to indicate whether or not the following constraint is satisfied:
\[
2x_1 + 3x_2 \leq 1.
\]

\((x_1 \text{ and } x_2 \text{ are non-negative continuous variables which cannot exceed } 1.\)\)

We wish to impose the following conditions:
\[
\delta = 1 \rightarrow 2x_2 + 3x_2 \leq 1, \tag{23}
\]
\[
2x_1 + 3x_2 \leq 1 \rightarrow \delta = 1. \tag{24}
\]

Using (12) \( M \) may be taken as 4 \((-2 + 3 - 1)\). This gives the following constraint representation of (23):
\[
2x_1 + 3x_2 + 4\delta \leq 5. \tag{25}
\]

Using (17) \( m \) may be taken as \(-1 \((-0 + 0 - 1)\). We will take \( \epsilon \) as 0–1. This gives the following constraint representation of (24):
\[
2x_1 + 3x_2 + 1 \cdot \epsilon \geq 1. \tag{26}
\]

The reader should verify that (25) and (26) have the desired effect by substituting 0 and 1 for \( \delta \).

In all the constraints derived in this section it is computationally desirable to make \( m \) and \( M \) as realistic as possible.

### 9.2 Logical Conditions and Zero–One Variables

In Section 9.1 it was pointed out that 0–1 variables are often introduced into an LP (or sometimes an IP) model as decision variables or indicator variables. Having introduced such variables it is then possible to represent logical connections between different decisions or states by linear constraints involving these 0–1 variables. It is at first sight rather surprising that so many different types of logical condition can be imposed in this way.

Some typical examples of logical conditions which can be so modelled are given below. Further examples are given by Williams (1977).

(a) If no depot is sited here then it will not be possible to supply any of the customers from the depot.

(b) If the library's subscription to this journal is cancelled then we must retain at least one subscription to another journal in this class.

(c) If we manufacture product A we must also manufacture product B or at least one of products C and D.

(d) If this station is closed then both branch lines terminating at the station must also be closed.

(e) No more than five of the ingredients in this class may be included in the blend at any one time.

(f) If we do not place an electronic module in this position then no wires can connect into this position.

(g) Either operation A must be finished before operation B starts or vice versa.

It will be convenient to use some notation from Boolean algebra in this section. This is the so-called set of *connectives* given below:

\[-\rightarrow \text{ means 'not'}.
\]
\[-\rightarrow \text{ means 'implies' or 'if... then'}.
\]
\[-\rightarrow \text{ means 'if and only if'}.
\]

These connectives are used to connect propositions denoted by \( P, Q, R, \text{etc.}, \)
\(-x > 0, x = 0, \delta = 1, \text{etc.}\)

For example, if \( P \) stands for the proposition 'I will miss the bus' and \( Q \) stands for the proposition 'I will be late', then \( P \rightarrow Q \) stands for the proposition 'If I miss the bus then I will be late'. \(-P \) stands for the proposition 'I will not miss the bus'.

As another example suppose that \( X_i \) stands for the proposition 'Ingredient \( i \) is in the blend' \((i \text{ ranges over the ingredients A, B and C})\). Then \( X_i \rightarrow (X_B \vee X_C) \) stands for the proposition 'If ingredient A is in the blend then ingredient B or C (or both) must also be in the blend'. This expression could also be written as \((X_i \rightarrow X_B) \vee (X_i \rightarrow X_C)\).

It is possible to define all these connectives in terms of a subset of them. For example they can all be defined in terms of the set \( \{ \lor, \rightarrow \} \). Such a subset is known as a complete set of connectives. We do not choose to do this and will retain the flexibility of using all the connectives listed above. It is important, however, to realize that certain expressions are equivalent to expressions involving other connectives. We give all the equivalences below which are sufficient for our purpose.

To avoid unnecessary brackets we will consider the symbols \( \sim, \cdot, \lor, \rightarrow \) each being more binding than their successor when written in this order. For example

\[
(P \cdot Q) \lor R \text{ can be written as } P \cdot (Q \lor R)
\]
\[
P \rightarrow (Q \lor R) \text{ can be written as } P \rightarrow Q \lor R
\]
\[\sim \sim P \text{ is the same as } P \]
$P \rightarrow Q$ is the same as $\neg P \lor Q$; \hspace{1cm} (2)

$P \rightarrow Q, R$ is the same as $(P \rightarrow Q) \land (P \rightarrow R)$ \hspace{1cm} (3)

$P \lor Q \land R$ is the same as $(P \lor Q) \land (P \lor R)$ \hspace{1cm} (4)

$P \land Q \rightarrow R$ is the same as $(P \rightarrow R) \lor (Q \rightarrow R)$ \hspace{1cm} (5)

$P \lor Q \rightarrow R$ is the same as $(P \lor R) \lor (Q \lor R)$ \hspace{1cm} (6)

$\neg (P \lor Q)$ is the same as $\neg P \land \neg Q$ \hspace{1cm} (7)

$\neg (P \land Q)$ is the same as $\neg P \lor \neg Q$ \hspace{1cm} (8)

(7) and (8) are sometimes known as De Morgan's laws.

Although Boolean algebra provides a convenient means of expressing and manipulating logical relationships, our purpose here is to express these relationships in terms of the familiar equations and inequalities of mathematical programming. (In one sense we are performing the opposite process to that used in the pseudo-Boolean approach to 0–1 programming mentioned in Section 8.3.)

We will suppose that indicator variables have already been introduced in the manner described in Section 9.1 to represent the decisions or states which we want logically to relate.

It is important to distinguish propositions and variables at this stage. We will use $X_i$ to stand for the proposition $\delta_i = 1$ where $\delta_i$ is a 0–1 indicator variable. The following propositions and constraints can easily be seen to be equivalent:

$X_1 \lor X_2$ is equivalent to $\delta_1 + \delta_2 \geq 1$; \hspace{1cm} (9)

$X_1 \land X_2$ is equivalent to $\delta_1 = 1, \delta_2 = 1$; \hspace{1cm} (10)

$\neg X_1$ is equivalent to $\delta_1 = 0$ (or $1 - \delta_1 = 1$); \hspace{1cm} (11)

$X_1 \rightarrow X_2$ is equivalent to $\delta_1 - \delta_2 \leq 0$; \hspace{1cm} (12)

$X_1 \leftarrow X_2$ is equivalent to $\delta_1 - \delta_2 = 0$. \hspace{1cm} (13)

To illustrate the conversion of a logical condition into a constraint we will consider an example.

**Example 1. Manufacturing**

If either of products A or B (or both) are manufactured, then at least one of products C, D, or E must also be manufactured.

Let $X_i$ stand for the proposition 'Product $i$ is manufactured' ($i$ is A, B, C, D, or E). We wish to impose the logical condition

$$(X_A \lor X_B) \rightarrow (X_C \lor X_D \lor X_E).$$ \hspace{1cm} (14)

Indicator variables are introduced to perform the following functions: $\delta_i = 1$ if

and only if product $i$ is manufactured; $\delta = 1$ if the proposition $X_A \lor X_B$ holds.

The proposition $X_A \lor X_B$ can be represented by the inequality

$$\delta_A + \delta_B \geq 1.$$ \hspace{1cm} (15)

The proposition $X_C \lor X_D \lor X_E$ can be represented by the inequality

$$\delta_C + \delta_D + \delta_E \geq 1.$$ \hspace{1cm} (16)

Firstly we wish to impose the following condition:

$$\delta_A + \delta_B \geq 1 \rightarrow \delta = 1.$$ \hspace{1cm} (17)

Using (19) of Section 9.1 we impose this condition by the constraint

$$\delta_A + \delta_B - 2\delta \leq 0.$$ \hspace{1cm} (18)

Secondly we wish to impose the condition

$$\delta = 1 \rightarrow \delta_C + \delta_D + \delta_E \geq 1.$$ \hspace{1cm} (19)

Using (18) of Section 9.1 this is achieved by the constraint

$$\neg \delta_C - \delta_D - \delta_E \leq 0.$$ \hspace{1cm} (20)

Hence the required extra condition can be imposed on the original model (LP or IP) by the following:

(i) Introduce 0–1 variables $\delta_A$, $\delta_B$, $\delta_C$, $\delta_D$, and $\delta_E$ and link them to the original (probably continuous) variables by constraints of type (1) and (7) of Section 9.1. It is not strictly necessary to include constraints of type (7) for the variables $\delta_A$ and $\delta_B$ since it is not necessary to have the conditions (4) of Section 9.1 in these cases.

(ii) Add the additional constraints (18) and (20) above.

This is not the only way to model this logical condition. Using the Boolean identity (6) above it is possible to show that condition (14) can be re-expressed as

$$[X_A \rightarrow (X_C \lor X_D \lor X_E)] \land [X_B \rightarrow (X_C \lor X_D \lor X_E)].$$ \hspace{1cm} (21)

The reader should verify that an analysis similar to that above results in the constraint (20) together with the following two constraints in place of (18):

$$\delta_A - \delta \leq 0;$$ \hspace{1cm} (22)

$$\delta_B - \delta \leq 0.$$ \hspace{1cm} (23)

Both ways of modelling the condition are correct. There are computational advantages in (22) and (23) over (18). This is discussed further in Section 10.1 of Chapter 10.

It is sometimes suggested that polynomial expressions in 0–1 variables are useful for expressing logical conditions. Such polynomial expressions can always be replaced by linear expressions with linear constraints, possibly with
a considerable increase in the number of 0–1 variables. For example the constraint
\[ \delta_1 \delta_2 = 0 \] (24)
represents the condition
\[ \delta_1 = 0 \lor \delta_2 = 0. \] (25)
More generally, if a product term such as \( \delta_1 \delta_2 \) were to appear anywhere in a model the model could be made linear by the following steps:
(i) Replace \( \delta_1 \delta_2 \) by a 0–1 variable \( \delta_3 \).
(ii) Impose the logical condition
\[ \delta_3 = 1 \iff \delta_1 = 1 \land \delta_2 = 1 \] (26)
by means of the extra constraints
\[ \begin{align*}
-\delta_1 + \delta_3 & \leqslant 0, \\
-\delta_2 + \delta_3 & \leqslant 0, \\
\delta_1 + \delta_2 - \delta_3 & \leqslant 1.
\end{align*} \] (27)
An example of the need to linearize products of 0–1 variables in this way arises in the DECENTRALIZATION problem in Part 3. Products involving more than two variables can be progressively reduced to single variables in a similar manner.
It is even possible to linearize terms involving a product of a 0–1 variable with a continuous variable. For example the term \( x \delta \), where \( x \) is continuous and \( \delta \) is 0–1, can be treated in the following way:
(i) Replace \( x \delta \) by a continuous variable \( y \).
(ii) Impose the logical conditions
\[ \begin{align*}
\delta = 0 & \iff y = 0, \\
\delta = 1 & \iff y = x
\end{align*} \] (28)
by the extra constraints
\[ \begin{align*}
y - M \delta & \leqslant 0, \\
-x + y & \leqslant 0, \\
x - y + M \delta & \leqslant M.
\end{align*} \] (29)
where \( M \) is an upper bound for \( x \) (and hence also \( y \)).
Other non-linear expressions (such as ratios of polynomials) involving 0–1 variables can also be made linear in similar ways. Such expressions tend to occur fairly rarely and are not therefore considered further. They do, however, provide interesting problems of logical formulation using the principles described in this section and can provide useful exercises for the reader.
The purpose of this section together with Example 1 has been to demonstrate a method of imposing logical conditions on a model. This is by no means the only way of approaching this kind of modelling. Different rule of thumb methods exist for imposing the desired conditions. Experienced modellers may feel that they could derive the constraints described here by easier methods. It has been the author’s experience, however that:
(i) Many people are unaware of the possibility of modelling logical conditions with 0–1 variables.
(ii) Among those people who realize that this is possible many find themselves unable to capture the required restrictions by 0–1 variables with logical constraints.
(iii) It is very easy to model a restriction incorrectly. By using concepts from Boolean algebra and approaching the modelling in the above manner it should be possible satisfactorily to impose the desired logical conditions.
A system for automating the formulation of logical conditions within standard predicates and implemented within the language PROLOG is described by McKinnon and Williams (1989).

9.3 Special Ordered Sets of Variables
Two very common types of restriction arise in mathematical programming problems for which the concept special ordered set of type 1 (SOS1) and special ordered set of type 2 (SOS2) have been developed. This concept is due to Beale and Tomlin (1969).
An SOS1 is a set of variables (continuous or integer) within which exactly one variable must be non-zero.
An SOS2 is a set of variables within which at most two can be non-zero. The two variables must be adjacent in the ordering given to the set.
It is perfectly possible to model the restrictions that a set of variables belongs to an SOS1 set or an SOS2 set using integer variables and constraints. The way in which this can be done is described below. There is great computational advantage to be gained, however, from treating these restrictions algorithmically. The way in which the branch and bound algorithm can be modified to deal with SOS1 and SOS2 sets is beyond the scope of this book. It is described in Beale and Tomlin.
Some examples are given on how SOS1 sets and SOS2 sets can arise.

Example 1. Depot Siting
A depot can be sited at any one of the positions A, B, C, D, or E. Only one depot can be built.
If 0–1 indicator variables \( \delta_i \) are used to perform the following purpose: \( \delta_i = 1 \) if and only if the depot is sited at \( i \) (\( i \) is A, B, C, D, or E), then the set of variables \( \{ \delta_1, \delta_2, \delta_3, \delta_4, \delta_5 \} \) can be regarded as an SOS1 set.
The SOS1 condition together with the constraint
\[ \delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 = 1 \] guarantees integrality and it is not necessary to stipulate that the \( \delta_i \) be integral. Only if the sites have a natural ordering is there great advantage to be gained in the SOS formulation.

Example 2. Capacity Extension

The capacity \( C \) of a plant can be extended in discrete amounts by increasing levels of investment \( I \).

If the set of variables \( (\delta_0, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5) \) is regarded as an SOS1 set then we can model
\[ C = C_1 \delta_1 + C_2 \delta_2 + C_3 \delta_3 + C_4 \delta_4 + C_5 \delta_5, \] \[ I = I_1 \delta_1 + I_2 \delta_2 + I_3 \delta_3 + I_4 \delta_4 + I_5 \delta_5, \] \[ \delta_0 + \delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 = 1. \]

It is not necessary to treat the \( \delta_i \) as integer variables since the SOS1 condition together with (4) forces integrality. Conceptually it is important to regard a SOS set as an entity. We can then regard \( C \) as a quantity which is a discrete function of \( I \). This can be regarded as a generalization of a 0–1 variable to more than two discrete values. Such a generalization is often more useful than the conventional general integer variable.

Although the most common application of SOS1 sets is to modelling what would otherwise be 0–1 integer variables with a constraint such as (4), there are other applications.

The most common application of SOS2 sets is to modelling non-linear functions as described by the following example.

Example 3. Non-linear Functions

In Section 7.3 the concept of a separable set was introduced in order to make a piecewise linear approximation to a non-linear function of a single variable. Using the \( \lambda \)-convention for such a separable formulation we obtained the following convexity constraint:
\[ \lambda_1 + \lambda_2 + \cdots + \lambda_n = 1. \]
In addition, in order that the coordinates of \( x \) and \( y \) should lie on the piecewise linear curve in Figure 7.15, it was necessary to impose the following extra restriction:
\[ \text{At most two adjacent } \lambda \text{s can be non-zero.} \] Instead of approaching this restriction through separable programming with the danger of local rather than global optima as described in Section 7.2, we can use an SOS2. The restriction (6) need not be modelled explicitly. Instead we can say that the set of variables \( (\lambda_1, \lambda_2, \ldots, \lambda_n) \) is an SOS2.

The formulation of a non-linear function in Example 3 demands that the non-linear function be separable, i.e. the sum of non-linear functions of a single variable. It was demonstrated in Section 7.4 how models with non-separable functions may sometimes be converted into models where the non-linearities are all functions of a single variable. While this is often possible, it can be cumbersome, increasing the size of the model considerably as well as the computational difficulty. An alternative is to extend the concept of a SOS set to that of a chain of linked SOS sets. This has been done by Beale (1980). The idea is best illustrated by a further example.

Example 4. Non-linear Functions of Two or More Variables

Suppose \( z = g(x, y) \) is a non-linear function of \( x \) and \( y \).

We define a grid of values of \( (x, y) \) (not necessarily equidistant) and associate non-negative 'weightings' \( \lambda_{ik} \) with each point in the grid as shown in Figure 9.2.

If the values of \( (x, y) \) at the grid points are denoted by \( (X_k, Y_k) \) we can approximate the function \( z = g(x, y) \) by means of the following relations:
\[ x = \sum_i \sum_k X_i \lambda_{ik} \] \[ y = \sum_i \sum_k Y_k \lambda_{ik}, \] \[ z = \sum_i \sum_k g(X_i, Y_k) \lambda_{ik}, \] \[ \sum_i \sum_k \lambda_{ik} = 1. \]

In addition it is necessary to impose the following restriction on the \( \lambda \) variables:
\[ \text{At most four neighbouring } \lambda \text{s can be non-zero.} \] This last condition is clearly a generalization of an SOS2 set. We can impose condition (11) in the following way. Let
\[ \xi_i = \sum_k \lambda_{ik}, \quad \eta_i = \sum_k \lambda_{ik} \]
for all \( s, t, (\xi_1, \xi_2, \xi_3, \ldots) \) and \( (\eta_1, \eta_2, \eta_3, \ldots) \) are each taken as SOS2 sets. The SOS2 condition for the first set allows \( \lambda_i \) to be non-zero in at most two neighbouring rows in Figure 9.2. For the second set the SOS2 condition allows
\[ \lambda_i \text{ to be non-zero in at most two neighbouring columns. For example we might}
\text{ have } \xi_3 = 1/3, \xi_2 = 2/3, \eta_4 = 1/4, \eta_5 = 3/4. \]

The values of \( \xi \) and \( \eta \) above could arise from \( \lambda_{25} = 1/6, \lambda_{26} = 1/6, \lambda_{35} = 1/12, \lambda_{36} = 7/12, \) all other \( \lambda_i \) being zero. They could, however, also arise from other values of the \( \lambda_i \) e.g. \( \lambda_{25} = 1/4, \lambda_{26} = 1/12, \lambda_{36} = 2/3 \) with all other \( \lambda_i \) being zero.

In order to get round this non-uniqueness we can restrict the non-zero \( \lambda_i \) to vertices of a triangle (such as in the second instance above). A lengthy way of doing this is to impose the extra constraints:

\[ \xi_i = \sum \lambda_{i,k} \leq 0 \]

and treat the \( \xi_i \) as a further SOS2 set.

If, however, we are content to restrict the \( x \) (or \( y \)) to grid values (i.e. not interpolate in that direction) then the problem does not arise. Indeed we can also avoid introducing the sets \( \xi_i \) so long as within each set \( \lambda_{ik} \), with the same \( s \), the member which is non-zero has the same index \( k \). The sets \( \lambda_{ik} \) are then known as a chain of linked SOS sets as described by Beale (1980) and the restriction can be dealt with algorithmically.

Some of the problems presented in Part 2 can be formulated to take advantage of special ordered sets. In particular, DECENTRALIZATION and LOGIC DESIGN can exploit SOS1.

While it is desirable to treat SOS sets algorithmically if this facility exists in the package being used, the restrictions which they imply can be imposed using 0–1 variables and linear constraints. This is now demonstrated.

Suppose \((x_1, x_2, \ldots, x_n)\) is an SOS1 set. If the variables are not 0–1 we introduce 0–1 indicator variables \( \delta_1, \delta_2, \ldots, \delta_n \) and link them to the \( x_i \) variables in the conventional way by constraints:

\[ x_i - M \delta_i \leq 0, \quad i = 1, 2, \ldots, n, \]
\[ x_i - m_i \delta_i \geq 0, \]

where \( M_i \) and \( m_i \) are constant coefficients being upper and lower bounds respectively for \( x_i \).

The following constraint is then imposed on the \( \delta_i \) variables:

\[ \delta_1 + \delta_2 + \cdots + \delta_n = 1. \]

If the \( x_i \) variables are 0–1 we can immediately regard them as the \( \delta_i \) variables above and only need impose the constraint (15).

To model an SOS2 set using 0–1 variables is more complicated. Suppose \((\lambda_1, \lambda_2, \ldots, \lambda_n)\) is an SOS2 set. We introduce 0–1 variables \( \delta_1, \delta_2, \ldots, \delta_{n-1} \) together with the following constraints:

\[ \lambda_1 - \delta_1 \leq 0, \]
\[ \lambda_2 - \delta_1 - \delta_2 \leq 0, \]
\[ \lambda_3 - \delta_2 - \delta_3 \leq 0, \]
\[ \lambda_{n-1} - \delta_{n-2} - \delta_{n-1} \leq 0, \]
\[ \lambda_n - \delta_{n-1} \leq 0, \]

and

\[ \delta_1 + \delta_2 + \cdots + \delta_{n-1} = 1. \]

This formulation suggests the relationship between SOS1 and SOS2 sets since (17) could be dispensed with by regarding the \( \delta_i \) as belonging to an SOS1 set as long as the \( \delta_i \) each have an upper bound of 1.

### 9.4 Extra Conditions Applied to Linear Programming Models

Since the majority of practical applications of LP give rise to mixed integer programming models where extra conditions have been applied to an otherwise LP model, this subject will be considered further in this section. A number of the commonest applications will briefly be outlined.

**Disjunctive Constraints**

Suppose that for an LP problem we do not require all the constraints to hold simultaneously. We do, however, require at least one subset of constraints to hold. This could be stated as

\[ R_1 \lor R_2 \lor \ldots \lor R_n, \]
where \( R_i \) is the proposition 'The constraints in subset \( i \) are satisfied' and constraints \( 1, 2, \ldots, N \) form the subset in question. (1) is known as a disjunction of constraints.

Following the principles of Section 9.1 we will introduce \( N \) indicator variables \( \delta_i \) to indicate whether the \( R_i \) are satisfied. In this case it is only necessary to impose the conditions

\[
\delta_i = 1 \rightarrow R_i. \tag{2}
\]

This may be done by constraints of type (12) or (18) (Section 9.1) taken singly or together according to whether \( R_i \) are ' \( \leq \)' , ' \( \geq \)' , or ' \( = \)' constraints. We can then impose condition (1) by the constraint

\[
\delta_1 + \delta_2 + \cdots + \delta_N \geq 1. \tag{3}
\]

An alternative formulation of (1) is possible. This is discussed in Section 10.2 and is due to Jeroslow and Lowe (1984), who report promising computational results in Jeroslow and Lowe (1985).

A generalization of (1) which can arise is the condition

At least \( k \) of \((R_1, R_2, \ldots, R_N)\) must be satisfied. \tag{4}

This is modelled in a similar way but using the constraint below in place of (3):

\[
\delta_1 + \delta_2 + \cdots + \delta_N \geq k. \tag{5}
\]

A variation of (4) is the condition

At most \( k \) of \((R_1, R_2, \ldots, R_N)\) must be satisfied. \tag{6}

To model (6) using indicator variables \( \delta_i \) it is only necessary to impose the conditions

\[
R_i \rightarrow \delta_i = 1. \tag{7}
\]

This may be done by constraints of type (17) or (19) (Section 9.1) taken together or singly according to whether \( R_i \) is a ' \( \leq \)' , ' \( \geq \)' , or ' \( = \)' constraint. Condition (6) can then be imposed by the constraint

\[
\delta_1 + \delta_2 + \cdots + \delta_N \leq k. \tag{8}
\]

Disjunctions of constraints involve the logical connective 'or' and necessitate IP models.

It is worth pointing out that the connective 'and' can obviously be coped with through conventional LP since a conjunction of constraints simply involves a series of constraints holding simultaneously. In this sense one can regard 'and' as corresponding to LP and 'or' as corresponding to IP.

**Non-convex Regions**

As an application of disjunctive constraints we will show how restrictions corresponding to a non-convex region may be imposed using IP. It is well known that the feasible region of an LP model is convex. (Convexity is defined in Section 7.2 of Chapter 7.) There are circumstances, however, in non-linear programming problems where we wish to have a non-convex feasible region. For example, we will consider the feasible region ABCDEFG of Figure 9.3. This is a non-convex region bounded by a series of straight lines. Such a region may have arisen through the problem considered or represent a piecewise linear approximation to a non-convex region bounded by curves.

We may conveniently think of the region ABCDEFG as made up of the union of the three convex regions ABJO, ODH, and KFGO, as shown in Figure 9.3. The fact these regions overlap will not matter.

Region ABJO is defined by the constraints

\[
x_2 \leq 3,
\]
\[
x_1 + x_2 \leq 4. \tag{9}
\]

Region ODH is defined by the constraints

\[-x_1 + x_2 \leq 0,
\]
\[3x_1 - x_2 \leq 8. \tag{10}\]

Region KFGO is defined by the constraints

\[
x_2 \leq 1,
\]
\[
x_1 \leq 5. \tag{11}\]

We will introduce indicator variables \( \delta_1, \delta_2, \) and \( \delta_3 \) to use in the following
conditions:

\[ \delta_1 = 1 \rightarrow (x_2 \leq 3), \quad (x_1 + x_2 \leq 4), \] (12)

\[ \delta_2 = 1 \rightarrow (-x_1 + x_2 \leq 0), \quad (3x_1 - x_2 \leq 8), \] (13)

\[ \delta_3 = 1 \rightarrow (x_2 \leq 1), \quad (x_1 \leq 5). \] (14)

(12), (13), and (14) are respectively imposed by the following constraints:

\[ x_2 + \delta_1 \leq 4, \] (15)
\[ x_1 + x_2 + 5\delta_1 \leq 9, \]
\[ -x_1 + x_2 + 4\delta_2 \leq 4, \] (16)
\[ 3x_1 - x_2 + 7\delta_2 \leq 15, \]
\[ x_2 + 3\delta_3 \leq 4, \] (17)
\[ x_1 \leq 5. \]

It is now only necessary to impose the condition that at least one of the set (9), (10), or (11) must hold. This is done by the constraint

\[ \delta_1 + \delta_2 + \delta_3 \geq 1. \] (18)

It would also be possible to cope with a situation in which the feasible region was disconnected in this way.

There is an alternative formulation for a connected non-convex region, such as that above, so long as the line joining the origin to any feasible point lies entirely within the feasible region. Seven 'weighting' variables \( \lambda_A, \lambda_B, \ldots, \lambda_G \) are associated with the vertices A, B, ..., G and incorporated in the following constraints:

\[ \lambda_B + 2\lambda_C + 4\lambda_D + 3\lambda_E + 5\lambda_F + 5\lambda_G - x_1 = 0, \] (19)
\[ 3\lambda_A + 3\lambda_B + 2\lambda_C + 4\lambda_D + \lambda_E + \lambda_F - x_2 = 0, \] (20)
\[ \lambda_A + \lambda_B + \lambda_C + \lambda_D + \lambda_E + \lambda_F + \lambda_G \leq 1. \] (21)

The \( \lambda \) variables are then restricted to form an SOS2. Notice that this is a generalization of the use of a SOS2 set to model a piecewise linear function such as that represented by the line ABCDEFG. In that case constraint (21) would become an equation, i.e. the \( \lambda \) would sum to 1. For the example here we simply relax this restriction to give a \( \leq \) constraint.

### Limiting the Number of Variables in a Solution

This is another application of disjunctive constraints. It is well known that in LP the optimal solution need never have more variables at a non-zero value than there are constraints in the problem. Sometimes it is required, however, to restrict this number still further (to \( k \)). To do this requires IP. Indicator variables \( \delta_i \) are introduced to link with each of the \( n \) continuous variables \( x_i \) in the LP problem by the condition

\[ x_i > 0 \rightarrow \delta_i = 1. \] (22)

As before this condition is imposed by the constraint

\[ x_i - M_i \delta_i \leq 0, \] (23)

where \( M_i \) is an upper bound on \( x_i \).

We then impose the condition that at most \( k \) of the variables \( x_i \) can be non-zero by the constraint

\[ \delta_1 + \delta_2 + \ldots + \delta_k \leq k. \]

A very common application of this type of condition is in limiting the number of ingredients in a blend. The FOOD MANUFACTURE 2 example of Part 1 is an example of this. Another situation in which the condition might arise is where it is desired to limit the range of products produced in a product mix type LP model.

### Sequentially Dependent Decisions

It sometimes happens that we wish to model a situation in which decisions made at a particular time will affect decisions made later. Suppose, for example, that in a multi-period LP model (\( n \) periods) we have introduced a decision variable \( y_t \) into each period to show how a decision should be made in each period. We will let \( y_t \) represent the following decisions: \( y_t = 0 \) means the depot should be permanently closed down; \( y_t = 1 \) means the depot should be temporarily closed (this period only); \( y_t = 2 \) means the depot should be used in this period. Clearly we would wish to impose (among others) the conditions

\[ y_t = 0 \rightarrow (y_{t+1} = 0), \quad (y_{t+2} = 0), \ldots, (y_n = 0). \] (24)

This may be done by the following constraints:

\[ -2y_1 + y_2 \leq 0, \]
\[ -2y_2 + y_3 \leq 0, \]
\[ -2y_{n-1} + y_n \leq 0. \] (25)

In this case the decision variable \( y_t \) can take three values. More usually it will be a \( 0-1 \) variable.

A case of sequentially dependent decisions arises in the MINING problem of Part 2.

### Economies of Scale

It was pointed out in Chapter 7 that economies of scale lead to a non-linear programming problem where the objective is equivalent to minimizing a non-convex function. In this situation it is not possible to reduce the problem
to an LP problem by piecewise linear approximations alone. Nor is it possible to rely on separable programming since local optima might result.

Suppose for example that we have a model where the objective is to minimize cost. The amount to be manufactured of a particular product is represented by a variable $x$. For increasing $x$ the unit marginal costs decrease.

Diagrammatically we have the situation shown in Figure 9.4. This may be the true cost curve or be a piecewise linear approximation.

The unit marginal costs are successively

$$\frac{c_1 - c_2}{b_1 - b_2} > \frac{c_2 - c_3}{b_2 - b_3} > \cdots$$

Using the $\lambda$-formulation of separable programming as described in Chapter 7, we introduce $n + 1$ variables $\lambda_i$ $(i = 0, 1, 2, \ldots, n)$ which may be interpreted as ‘weights’ attached to the vertices $A$, $B$, $C$, $D$, etc. We then have

$$x = b_0 \lambda_0 + b_1 \lambda_1 + \cdots + b_n \lambda_n,$$

$$\text{cost} = c_0 \lambda_0 + c_1 \lambda_1 + \cdots + c_n \lambda_n.$$  

The set of variables $\lambda_0, \lambda_1, \ldots, \lambda_n$ is now regarded as a special ordered set of type 2. If IP is used, a global optimal solution can be obtained.

It is also, of course, possible to model this situation using the $\delta$-formulation of separable programming.

**Discrete Capacity Extensions**

It is sometimes unrealistic to regard an LP constraint (usually a capacity constraint) as applying in all circumstances. In real life it is often possible to violate the constraint at a certain cost. This topic has already been mentioned in Section 3.3. There, however, we allowed this constraint to be relaxed continuously. Often this is not possible. Should the constraint be successively relaxed it may be possible that it can only be done in finite jumps, e.g. we buy in whole new machines or whole new storage tanks.

Suppose the initial right-hand side (RHS) value is $b_0$ and that this may be successively increased to $b_1, b_2, \ldots, b_n$.

We have

$$\sum_j a_j x_j \leq b_i;$$

also

$$\text{cost} = \begin{cases} 0 & \text{if } i = 0, \\ c_i & \text{otherwise}, \end{cases}$$

where $0 < c_1 < c_2 < \cdots < c_n$.

This situation may be modelled by introducing 0–1 variables $\delta_0, \delta_1, \delta_2, \ldots, \delta_n$, etc., to represent the successive possible RHS values applying. We then have

$$\sum_j a_j x_j - b_0 \delta_0 - b_1 \delta_1 - \cdots - b_n \delta_n \leq 0.$$  

The following expression is added to the objective function:

$$c_0 \delta_0 + c_1 \delta_1 + \cdots + c_n \delta_n.$$  

The set of variables $(\delta_0, \delta_1, \ldots, \delta_n)$ may be treated as an SOS1 set. If this is done then the $\delta_i$ can be regarded as continuous variables having a generalized upper bound of 1, i.e. the integrality requirement may be ignored.

**Maximax Objectives**

Suppose we had the following situation:

Maximize $\left( \text{Maximum} \left( \sum_j a_j x_j \right) \right)$

subject to conventional linear constraints.

This is analogous to the minimax objective discussed in Section 3.2 but unlike that case it cannot be modelled by linear programming.

We can, however, treat it as a case of a disjunctive constraint and use integer programming. The model can be expressed as:

Maximize $\sum_j a_j x_j - z$ or $\sum_j a_j x_j - z = 0$ or $\cdots$ etc.
CHAPTER 10

Building Integer Programming Models II

10.1 Good and Bad Formulations

Most of the considerations of Section 3.4 concerning linear programming (LP) models will also apply to integer programming (IP) models and will not be reconsidered here. There are, however, some important additional considerations which must be taken account of when building IP models. The primary additional consideration is the much greater computational difficulty of solving IP models over LP models. It is fairly common to build an IP model only to find the cost of solving it prohibitive. Frequently it is possible to reformulate the problem, giving another, easier to solve, model. Such reformulations must often be considered in conjunction with the solution strategy to be employed. It will be assumed throughout that the branch and bound method described in Section 8.3 of Chapter 8 is to be used.

In some respects there is much greater flexibility possible in building IP models than in building LP models. The flexibility results in a greater divergence between good and bad models of a practical problem. The purpose of this section is to suggest ways in which good models may be constructed.

It is convenient to consider variables and constraints separately. There is often the possibility of using many or few variables and many or few constraints in a model. The considerations governing this will be considered.

The Number of Variables in an IP Model

We will confine our attention here to the number of integer variables in an IP model since this is often regarded as a good indicator of the computational difficulty.

Suppose we had a 0–1 IP model (either mixed integer or pure integer). If the model had \( n \) 0–1 variables this would indicate \( 2^n \) possible settings for the variables and hence \( 2^n \) potential nodes hanging at the bottom of the solution tree. In total there would be \( 2^{n+1} - 1 \) nodes in such a tree. One might therefore expect the solution time to go up exponentially with the number of 0–1 variables. For quite modest values of \( n \), \( 2^n \) is very large, e.g., \( 2^{100} \) is greater than one million raised to the power 5. The situation is not of course anywhere as bad as this, since many of the \( 2^n \) potential nodes will never be examined.

The branch and bound method rules out large sections of the potential tree from examination as being infeasible or worse than solutions already known. It is, however, worth pausing to consider the fact that one may sometimes solve 100 0–1 variable IP problems in a few hundred nodes. This represents only about 0.00 ... 01 per cent of the potential total where there are 28 zeros after the decimal point. In view of this very surprising efficiency that the branch and bound method exhibits over the potential amount of computation, the number of 0–1 variables is often a very poor indicator of the difficulty of an IP model. We will, however, suggest one circumstance in which the number of such variables might be usefully reduced, later in this section. Before doing that we will indicate ways in which the number of integer variables in a model might be increased to good effect.

It is convenient here to describe a well known device for expanding any general integer variable in a model to a number of 0–1 variables. Suppose \( \gamma \) is a general (non-negative) integer variable with a known upper bound of \( u \) (an upper bound is required for all integer variables in an IP model if the branch and bond method is to be used), i.e.,

\[
0 \leq \gamma \leq u.
\]

\( \gamma \) may be replaced in this model by the expression

\[
\delta_0 + 2\delta_1 + 4\delta_2 + 8\delta_3 + \cdots + 2^k\delta_k,
\]

where the \( \delta_i \) are 0–1 variables and \( 2^k \) is the smallest power of 2 greater than or equal to \( u \).

It is easy to see that the expression (1) can take any possible integral value between 0 and \( u \) by different combinations of values for the \( \delta_i \) variables. Clearly the number of 0–1 variables required in an expansion like this is roughly \( \log_2 u \). In practice \( u \) will probably be fairly small and the number of 0–1 variables produced not too large. If, however, this device were employed on a lot of the variables in the model the result might be a great expansion in model size. Generally there will be little virtue in an expansion of this sort except to facilitate the use of some specialized algorithm applying only to 0–1 problems. This is beyond the scope of this book.

Although there is some virtue in keeping an LP model compact, any such advantages that this may imply for the corresponding IP model are usually drowned by other much more important considerations. Using the branch and bound method there is sometimes virtue in introducing extra 0–1 variables as useful variables in the branching process. Such 0–1 variables represent ‘dichotomies’ in the system being modelled. To make such dichotomies explicit can be valuable, as is demonstrated in the following example due to Jeffreys (1974).

Example 1

One new factory is to be built. The possible decisions are represented by 0–1
variables $\delta_{n,b}$, $\delta_{n,c}$, $\delta_{b,c}$, and $\delta_{s,c}$:

\[
\delta_{n,b} = \begin{cases} 
1 & \text{if the factory is in the north and uses} \\
0 & \text{a batch process,} \\
& \text{otherwise;}
\end{cases}
\]

\[
\delta_{n,c} = \begin{cases} 
1 & \text{if the factory is in the north and uses} \\
0 & \text{a continuous process,} \\
& \text{otherwise;}
\end{cases}
\]

\[
\delta_{b,c} = \begin{cases} 
1 & \text{if the factory is in the south and uses} \\
0 & \text{a batch process,} \\
& \text{otherwise;}
\end{cases}
\]

\[
\delta_{s,c} = \begin{cases} 
1 & \text{if the factory is in the south and uses} \\
0 & \text{a continuous process,} \\
& \text{otherwise.}
\end{cases}
\]

The condition that only one factory be built can be represented by the constraint

\[
\delta_{n,b} + \delta_{n,c} + \delta_{b,c} + \delta_{s,c} = 1. \tag{2}
\]

It is not possible, using constraint (2), to express the dichotomy 'either we site the factory in the north or we site it in the south' by a single 0–1 variable. Since this is clearly an important decision, it would be advantageous to have a 0–1 variable indicating the decision. By adding an extra 0–1 variable $\delta$ to represent this decision, together with the extra constraints

\[
\delta_{n,b} + \delta_{n,c} - \delta = 0, \tag{3}
\]

\[
\delta_{b,c} + \delta_{s,c} + \delta = 1, \tag{4}
\]

this is possible, $\delta$ is a valuable variable to have at our disposal since use can be made of it as a variable to branch on in the tree search. The dichotomy 'either we use a batch process or we use a continuous process' could also be represented by another 0–1 decision variable in a similar way.

Another use of extra integer variables in a model is to specify the slack variable in a constraint, made up of only integer variables, as itself being integer. For example, if all the variables, coefficients, and right-hand side are integer in the constraint

\[
\sum_j a_j x_j \leq b, \tag{5}
\]

we can put a slack variable $u$ and specify this variable to be integer, giving

\[
\sum_j a_j x_j + u = b. \tag{6}
\]

Normally such a slack variable would be inserted by the mathematical programming package used but treated only as a continuous variable. There is advantage in treating $u$ as an integer variable and giving it priority in the branching process. When $u$ is the variable branched on, constraint (6) will have the effect of a cutting plane and restrict the feasible region of the corresponding LP problem. This idea is due to Mitra (1973).

To summarize there is often advantage in increasing rather than decreasing the number of integer variables in a model especially if these extra variables are made use of in the tree search strategy. Such ideas can be used to advantage in some of the IP problems given in Part 2.

In some circumstances, however, there is advantage to be gained in reducing the number of integer variables. A case of this is illustrated in the following example. Here the problem exhibits a symmetry which can be computationally undesirable.

Example 2

As part of a larger IP model the following variables are introduced.

\[
\delta_{ij} = \begin{cases} 
1 & \text{if lorry } i \text{ is sent on trip } j \\
0 & \text{otherwise}
\end{cases}
\]

where $i = \{1, 2, 3\}$, $j = \{1, 2\}$. The lorries are indistinguishable in terms of running costs, capacities, etc.

Clearly corresponding to each possible integer solution, e.g.

\[
\delta_{11} = 1, \quad \delta_{21} = 1, \quad \delta_{31} = 1, \tag{7}
\]

there will be symmetric integer solutions, e.g.

\[
\delta_{12} = 1, \quad \delta_{22} = 1, \quad \delta_{32} = 1. \tag{8}
\]

As the branch and bound tree search progresses each symmetric solution may be obtained at a separate node.

A better formulation involving less integer variables and avoiding the symmetry could be devised using the following integer variables:

\[
n_i = \text{number of lorries sent on trip } j. \tag{9}
\]

The solutions (7) and (8) above would now be indistinguishable as the solution

\[
n_1 = 1, \quad n_2 = 2.
\]

The Number of Constraints in an IP Model

It was pointed out in Chapter 3 that the difficulty of an LP model is very dependent on the number of constraints. Here we will show that in an IP model this effect is often completely drowned by other considerations. In fact an IP model is often made easier to solve by expanding the number of constraints.

In an LP model we are searching for vertex solutions on the boundary of the
feasible region. For the corresponding IP model we are interested in integer points which may well lie in the interior of the feasible region. This is illustrated in Figure 10.1. ABCD is the feasible region of the LP problem but we must confine our attention to the lattice of integer points. For an IP model the corresponding LP model is known as the LP relaxation.

In this diagram we are supposing both \( x_1 \) and \( x_2 \) to be integer variables. For mixed integer problems we are interested in points in an analogous diagram to Figure 10.1, some of whose coordinates are integer, but where other coordinates are allowed to be continuous. Clearly this situation is difficult to picture in a few dimensions. Suppose, however, that there were a further continuous variable \( x_3 \) to be considered in Figure 10.1. This would give rise to a coordinate coming out at right angles to the page. Feasible solutions to the mixed integer problem would consist of lines parallel to the \( x_3 \) axis coming out of the page from the integer points in Figure 10.1. These lines would have, of course, to lie inside the three-dimensional feasible region of the corresponding LP problem.

Ideally we would like to reformulate the IP model so that the feasible region of the corresponding LP model became PQRSTUV. This is known as the convex hull of feasible integer points in ABCD. It is the smallest convex set containing all the feasible integer points. If it were possible to reformulate the IP problem in this way we could solve the problem as an LP problem since the integrality requirement would automatically be satisfied. Such a formulation, where the LP relaxation gives the convex hull of IP solutions, is known as sharp.

Each vertex (and hence optimal solution) of the new feasible region PQRSTUV is an integer point. Unfortunately in many practical problems the effort required to obtain the convex hull of integer points would be enormous and far outweigh the computation needed to solve the original formulation of the IP problem. There are, however, important classes of problems where:

(i) The straightforward formulation results in an IP model where the feasible region is already the convex hull of integer points.

(ii) The problem can fairly easily be reformulated to give a feasible region corresponding to the convex hull of integer points.

(iii) By reformulating it is possible to reduce the feasible region of the LP problem to nearer that of the convex hull of integer points.

We will consider each of these classes of problem in turn.

Case (i) concerns some problems which have already been considered in Section 5.3. Although superficially these problems might appear to give rise to PIP models, the optimal solution of the corresponding LP problem always results in integer values for the integer variables. It is not therefore necessary to treat the model as any other than an LP problem. Problems falling into this category include the transportation problem, the minimum cost network flow problem, and the assignment problem. It is sometimes possible to recognize when an IP model has a structure which guarantees that the corresponding LP model will have an integer optimal solution. Clearly it is very useful to be able to recognize this property since the high computational cost of integer programming need not be incurred. Consider the following LP model: maximize \( c^T x \), subject to \( Ax = b \) and \( x \geq 0 \).

Here we assume that slack variables have been added to the constraints, if necessary, to make them all equalities. For the above model to yield an optimal solution with all variables integer for every objective coefficient vector \( c \) and integer right-hand side \( b \) the matrix \( A \) must have a property known as total unimodularity.

**Definition**

A matrix \( A \) is totally unimodular if every square sub-matrix of \( A \) has its determinant equal to \( 0 \) or \( \pm 1 \).

The fact that this property guarantees that there will be an integer optimal solution to the LP model (for any \( c \) and integer \( b \)) is proved in Garfinkel and Nemhauser (1972, p. 67). Unfortunately the above definition of total unimodularity is of little help in detecting the property. To evaluate the determinant of every square sub-matrix would be prohibitive. There is, however, a property (which we call \( P \)) which is easier to detect and which guarantees total unimodularity. It is, however, only a sufficient condition, not a necessary condition. Matrices without property \( P \) may still be totally unimodular.
Property $P$

1. Each element of $A$ is 0, 1 or $-1$.
2. No more than two non-zero elements appear in each column.
3. The rows can be partitioned into two subsets $P_1$ and $P_2$ such that: (a) if a column contains two non-zero elements of the same sign, one element is in each of the subsets; (b) if a column contains two non-zero elements of opposite sign, both elements are in the same subset.

A particular case of the property $P$ is when subset $P_1$ is empty and $P_2$ consists of all the rows of $A$. Then for the property to hold we must have all columns consisting of either one non-zero element ±1 or two non-zero elements +1 and $-1$.

As an example of this property holding we can consider the small transportation problem of Section 5.5:

$$
\begin{align*}
-x_{11} - x_{13} - x_{14} - x_{15} &= -135, \\
-x_{12} - x_{14} - x_{15} &= -56, \\
-x_{21} - x_{22} - x_{23} - x_{25} &= 93, \\
-x_{31} - x_{32} - x_{33} - x_{34} - x_{35} &= -9, \\
x_{11} + x_{12} + x_{13} &= 62, \\
x_{12} + x_{13} &= 83, \\
x_{13} + x_{14} &= 39, \\
x_{14} + x_{15} &= 91, \\
x_{15} + x_{25} + x_{35} &= 9.
\end{align*}
$$

Each column clearly contains $a + 1$ and $a - 1$, showing the property to hold and hence guaranteeing total unimodularity. There is often virtue in trying to reformulate a model in order to try to capture this easily detected property. The automation of such reformulation (where possible) was described in Section 5.4. In case (ii) below the dual situation is illustrated.

Although the property above refers to a partitioning of the rows of a matrix $A$ into two subsets, the partitioning could equally well apply to the columns. If $A$ is totally unimodular, its transpose $A^T$ must also be totally unimodular. Again a particular instance of this property guaranteeing total unimodularity is when each row of the matrix $A$ of an LP model contains $a + 1$ and $a - 1$.

Finally it should be pointed out that total unimodularity is a strong property that guarantees integer optimal solutions to an LP problem for all $c$ and integer $b$. Many IP models for which the matrix $A$ is not totally unimodular frequently (although not always) produce integer solutions to the optimal solution of the corresponding LP problem. In particular this often happens with the set packing, partitioning, and covering problems discussed in Section 9.5. There are good reasons why this is likely to happen for these types of problem. Such considerations are, however, fairly technical and beyond the scope of this book. They are discussed in Chapter 8 of Garfinkel and Nemhauser (1972). Properties of $A$ which guarantee integer LP solutions for a specific right-hand side $b$ are discussed by Padberg (1974) for the case of the set partitioning problem.

The discussion of total unimodularity above applies only to PIP models. Clearly there are corresponding considerations for MIP models, where integer values for the integer variables in the optimal LP solution are guaranteed. Such considerations are, however, very difficult and there is little theoretical work as yet which is of value to practical model builders.

Case (ii) above concerns problems where, with a little thought, a reformulation can result in a model with the total unimodularity property. Consider a generalization of the constraint (18) of Section 9.2:

$$\delta_1 + \delta_2 + \ldots + \delta_n - n\delta \leq 0,$$

where $\delta_1$ and $\delta$ are $0$-$1$ variables.

This kind of constraint arises fairly frequently in IP models and represents the logical condition

$$\delta_1 = 1 \lor \delta_2 = 1 \lor \ldots \lor \delta_n = 1 \rightarrow \delta = 1.$$  

Sometimes this condition is more easily thought of as the logical equivalent condition

$$\delta = 0 \rightarrow \delta_1 = 0 \land \delta_2 = 0 \land \ldots \land \delta_n = 0.$$  

It was shown in Section 9.2 that by a different argument constraint (18) of that section can be reformulated using two constraints. A similar reformulation can be applied here, giving the $n$ constraints

$$\delta_1 - \delta \leq 0,$$

$$\delta_2 - \delta \leq 0,$$

$$\ldots,$$

$$\delta_n - \delta \leq 0.$$  

Should all the constraints in the model be similar to the constraints of (14) then the dual problem has the property $P$ described above which guarantees total unimodularity. There is, therefore, great virtue in such a reformulation since the high computational costs associated with an IP problem over an LP problem is avoided. An example of a reformulation of a problem in this way is described by Rhys (1970). He also demonstrates another advantage of the reformulation as yielding a more meaningful economic interpretation of the shadow prices. This topic is considered in Section 10.3. A practical example of a formulation such as that described above is given in Part 3 where the formulation of the OPENCAST MINING problem is discussed.

Constraint (11) above shows the possibility of sometimes reformulating a PIP problem that is not totally unimodular in order to make it totally unimodular. There is also virtue in reformulating an already totally unimodular problem which we do not know to be totally unimodular if by so doing we convert it into a form where property $P$ applies. Examples of this are given by Veinott and Wagner (1962) and Daniel (1973). As with case (i) the above
discussion of case (ii) only applies to PIP problems. Again it must be possible (although difficult) to generalize these ideas to MIP problems.

Case (iii) concerns problems where there is either no obvious totally unimodular reformulation or where the problem gives a MIP model. In cases (i) and (ii) we were reducing the feasible region to the convex hull of feasible integer points, even though this was not obvious from the algebraic treatment given. It is sometimes possible to go part way towards this aim. Suppose for example (as might frequently happen in a MIP model) that only some of the constraints were of the form (11). By expanding these constraints into the series of constraints (14) we would reduce the size of the LP feasible region. Even though the existence of other constraints in the problem might result in some integer variables taking fractional values in the LP optimal solution this solution should be 'nearer' the integer optimal solution than would be the case with the original model. The term 'nearer' is purposely vague. A reformulation such as this might result in the objective value at node 1 of the solution tree (for example Figure 8.1) being closer to the objective value of the optimal integer solution when found. On the other hand, it might result in there being less fractional solution values in the LP optimum. Whatever the result of the reformulation one would normally expect the solution time for the reformulated model to be less than for the original model. Constraints involving just two coefficients +1 and -1 also arise in models involving sequentially dependent decisions as described in Section 9.4. Such constraints are always to be desired even if their derivation results in an expansion of the constraints of a model. An example of this is the suggested formulation in Part 3 for the MINING problem.

It is worth indicating in another way why a series of constraints such as (14) is preferable to the single constraint (11). Although (11) and (14) are exactly equivalent in an IP sense they are certainly not equivalent in an LP sense. In fact (11) is the sum of all the constraints in (14). By adding together constraints in an LP problem one generally weakens their effect. This is what happens here. (11) admits fractional solutions which (14) would not admit. For example the solution

\[ \delta_1 = \frac{1}{2}, \quad \delta_2 = \delta_3 = \frac{1}{4}, \quad \delta = \frac{1}{n}, \quad \text{all other } \delta_i = 0 \tag{15} \]

satisfies (11) but breaks (14) (for \( n \geq 3 \)).

Hence (14) is more effective at ruling out unwanted fractional solutions.

The ideas discussed above are relevant to the FOOD MANUFACTURE 2 problem which gives rise to a MIP model and to the DECENTRALIZATION and LOGICAL DESIGN problems which give rise to PIP models.

Some of the material so far presented in this section was first published by Williams (1974). A discussion of some very similar ideas applied to a more complicated version of the DECENTRALIZATION problem is given in Beale and Tomlin (1972).

It is also relevant here to discuss the value of the coefficient 'M' when linking indicator variables to continuous variables by constraints such as (1), (12), (19), (21) of Section 9.1. These types of constraints usually (but not always) arise in MIP models.

We will consider the simplest way in which such a constraint arises when we are using a 0–1 variable \( \delta \) to indicate the condition below on the continuous variable \( x \):

\[ x > 0 \rightarrow \delta = 1. \tag{16} \]

This condition is represented by the constraint

\[ x - M\delta \leq 0. \tag{17} \]

So long as \( M \) is a true upper bound for \( x \) condition (16) is imposed, however large we make \( M \). There is virtue, however, in making \( M \) as small as possible without imposing a spurious restriction on \( x \). This is because by making \( M \) smaller we reduce the size of the feasible region of the LP problem corresponding to the MIP problem. Suppose, for example, we took \( M \) as 1000 when it was known that \( x \) would never exceed 100. The following fractional solution would satisfy (17):

\[ x = 70, \quad \delta = \frac{1}{2}. \tag{18} \]

but would violate (17) if \( M \) were taken as 100. There are other good reasons for making \( M \) as realistic as possible. For example, if \( M \) were again taken as 1000 the following fractional solution would satisfy (17):

\[ x = 5, \quad \delta = 0.005. \tag{19} \]

A small value of \( \delta \) such as this might well fall below the tolerance which indicates whether a variable were integer or not. If it did \( \delta \) would be taken as 0 giving the spurious integer solution

\[ x = 5, \quad \delta = 0. \tag{20} \]

If, however, \( M \) was made smaller this would be less likely to happen. Finally, the inadvisability of having coefficients of widely differing magnitudes, as mentioned in Section 3.4, makes a small value of \( M \) desirable.

It is also sometimes possible to split up a constraint using a coefficient \( M \) in an analogous fashion to the way in which (11) was split up into (14). This is demonstrated by the following example.

**Example 3**

\[ \delta = \begin{cases} 1 & \text{if the depot is built,} \\ 0 & \text{otherwise.} \end{cases} \]

If the depot is built it can supply customer \( i \) with a quantity up to \( M_i, i = 1, 2, \ldots, n \). If the depot is not built, none of these customers can be supplied with anything.

\[ x_i = \text{quantity supplied to customer } i. \]
These conditions can be imposed by the following constraint:

\[ x_1 + x_2 + \ldots + x_n - M\delta \leq 0, \]  \hspace{1cm} (21)

where \( M = M_1 + M_2 + \ldots + M_n \).

On the other hand, the following constraints are superior as the corresponding LP problem is more constrained:

\[ \begin{align*}
  x_1 - M_1\delta & \leq 0, \\
  x_2 - M_2\delta & \leq 0, \\
  & \vdots \\
  x_n - M_n\delta & \leq 0.
\end{align*} \hspace{1cm} (22)

To summarize this section the main objectives of an IP formulation should be as follows:

1. To use integer variables which can be put to a good purpose in the branching process of the branch and bound method. If necessary, introduce extra 0–1 variables to create meaningful dichotomies.
2. To make the LP problem corresponding to the IP problem as constrained as possible.

A final objective not yet mentioned in this section is

3. To use special ordered sets as described in Section 9.3 if it is possible and the computer package used is capable of dealing with them.

Further ways of reformulating IP models in order to ease their solution are described in the next section.

Before a large IP model is built it is often a very good idea to build a small version of the model first. Experimentation with different solution strategies and possibly with reformulation can give valuable experience before embarking on the much larger model.

Sometimes, by examining the structure of a model, it is possible to make observations that lead one to a tightening of the constraints. A dramatic example of this (even when the application was unknown) is described by Daniel (1978), resulting in the solution of a reformulated model in 171 nodes where previously the tree search had been abandoned after 4757 nodes.

The automatic reformulation of IP models in order to tighten the LP relaxation is described by Crowder, Johnson, and Padberg (1983) and Van Roy and Wolsey (1984).

### 10.2 Simplifying an Integer Programming Model

In the last section it was shown that it is often possible to reformulate an IP model in order to create another model which is easier to solve. This is sometimes made possible by considering the practical situation being modelled. In this section we will be concerned with rather less obvious transformation of an IP model. Again the aim will be to make the model easier to solve.

**Tightening Bounds**

In Section 3.4, part of the procedure of Brealey, Mitra, and Williams (1975) for simplifying LP models was outlined. The full procedure of that procedure involves removing redundant simple bounds in an LP model. It is not, however, generally worthwhile removing redundant bounds on an integer variable. Instead it is better to tighten the bounds if possible. The argument for doing this is similar to some of the reformulation arguments used in the last section. By tightening bounds the corresponding LP problem may be made more constrained resulting in the optimal solution to the LP problem being closer to the optimal IP solution. In order to illustrate the procedure a small example from Balas (1965) will be used. This example was also used in the description of the procedure given by Brealey, Mitra, and Williams.

**Example 1**

Minimize \[ 5\delta_1 + 7\delta_2 + 10\delta_3 + 3\delta_4 + \delta_5 \]

subject to

\[ \begin{align*}
  \delta_1 - 3\delta_2 + 5\delta_3 + \delta_4 - \delta_5 & \geq 2, \\
  -2\delta_1 + 6\delta_2 - 3\delta_3 - 2\delta_4 + 2\delta_5 & \geq 0, \\
  -\delta_2 + 2\delta_3 - 2\delta_4 - \delta_5 & \geq 1.
\end{align*} \hspace{1cm} (R1) \]

The \( \delta_i \) are all 0–1 variables.

1. By constraint (R3)

\[ 2\delta_3 \geq 1 + \delta_2 + \delta_4 + \delta_5 \geq 1. \]

Hence

\[ \delta_3 \geq \frac{1}{2}. \]

Since \( \delta_3 \) is an integer variable this implied lower bound may be tightened. In this case (since \( \delta_3 \) is 0–1) \( \delta_3 \) may be set to 1 and removed from the problem.

2. By constraint (R2)

\[ 6\delta_2 \geq 3 + 2\delta_1 + 2\delta_4 - 2\delta_5 \geq 1. \]

Hence

\[ \delta_2 \geq \frac{1}{6}. \]

Similarly the lower bound of \( \delta_2 \) may be tightened to 1 so fixing \( \delta_2 \) at 1.

3. By constraint (R3)

\[ \delta_4 \leq -\delta_5 \leq 0. \]
Hence
\[ \delta_0 \leq 0. \]

Therefore \( \delta_0 \) can be fixed at 0.

(4) By constraint (R3), \( \delta_3 \leq 0 \). Therefore \( \delta_3 \) can be fixed at 0.

(5) All the remaining constraints now turn out to be redundant and may be removed. The only remaining variable is \( \delta_1 \) which must obviously be set to 0.

This example is obviously an extreme case of the effect of tightening bounds in an IP model since this procedure alone completely solves the problem.

**Simplifying a Single Integer Constraint to Another Single Integer Constraint**

Consider the integer constraint
\[ 4\gamma_1 + 6\gamma_2 \leq 9, \]

where \( \gamma_1 \) and \( \gamma_2 \) are general integer variables. By looking at this constraint geometrically in Figure 10.2 it is easy to see that it may also be written as
\[ \gamma_1 + 2\gamma_2 \leq 2, \]

In Figure 10.2 the original constraint (1) indicates the feasible points must lie to the left of AB. By shifting the line AB to CD no integer points are excluded from, and no new integer points are included in, the feasible region. CD gives rise to the new constraint (2).

Clearly there are advantages in using (2) rather than (1) since the feasible region of the corresponding LP problem has been reduced. While constraints such as (1) involving general integer variables do not arise very frequently such constraints can occur involving only 0–1 variables. We will therefore confine our attention to the 0–1 case. Should more general integer variables be involved it is, of course, always possible to expand them into 0–1 variables as described in Section 10.1. Our problem will now be, given a constraint such as
\[ a_1\delta_1 + a_2\delta_2 + \cdots + a_n\delta_n \leq a_0, \]

where the \( \delta_i \) are 0–1 variables, to re-express it as an equivalent constraint
\[ b_1\delta_1 + b_2\delta_2 + \cdots + b_n\delta_n \leq b_0, \]

where (4) is more constrained than (3) in the corresponding LP problem. There is no loss of generality in assuming all the coefficients of (3) and (4) to be non-negative since should a negative coefficient \( a_i \) occur in (3) the corresponding variable \( \delta_i \) may be complemented by the substitution
\[ \delta_i = 1 - \delta_i, \]

making the new coefficient of \( \delta_i \) positive.

Clearly '\( \geq \)' constraints can be converted to '\( \leq \)'. Equality constraints are most conveniently dealt with by converting them into a '\( \leq \)' together with a '\( \geq \)' constraint. The result will be two simplified constraints in place of the original single '\( = \)' constraint.

The simplification of a pure 0–1 constraint such as (3) in order to produce (4) can itself be formulated and solved as an LP problem. Rather than present the technical details of the procedure here a specific problem, OPTIMIZING A CONSTRAINT is given in Part 2. The formulation and discussion in Part 3 should make the general procedure clear.

If the original constraint to be simplified involved general integer, rather than 0–1 integer, variables and it is desired to replace it by a constraint in the same variables, then it will be necessary to restrict the new coefficients of the 0–1 form. For example, suppose we had a general integer variable \( \gamma \) (\( \leq 7 \)). If its original coefficient were \( a \) in the 0–1 form of the constraint, we would have the term
\[ a\gamma_1 + 2a\gamma_2 + 4a\gamma_3, \]

with \( \gamma_1 + 2\gamma_2 + 4\gamma_3 \) representing variable \( \gamma \), where \( \gamma_1, \gamma_2 \) and \( \gamma_3 \) are 0–1 variables. The resultant simplification would produce an equivalent term
\[ b_1\gamma_1 + 2b_2\gamma_2 + 4b_3\gamma_3. \]

In order for this term to be replaceable by a term \( b\gamma \) we would have to ensure that
\[ b_1 = 2b_2, \quad b_2 = 2b_1, \]
giving \( b = b_1 \).

Using the LP formulation described for the 0–1 example in Part 2 it would be necessary to impose conditions such as (7) as extra LP constraints for the general integer case.

The effort of trying to simplify single integer constraints in this way will
often not be worthwhile. In many models these constraints represent logical conditions and are already in their simplest form as single constraints. Applications where such simplification could possibly prove worthwhile are project selection and capital budgeting problems. It also proves worthwhile to simplify single constraints in this way in the MARKET SHARING problem presented in Part 2.

The simplification of a single 0–1 constraint into another single 0–1 constraint considered here has been described by Bradley, Hammer, and Wolsey (1974).

Simplifying a Single Integer Constraint to a Collection of Integer Constraints

We will again confine our attention to pure 0–1 constraints given that general integer variables can be expanded, if necessary, into 0–1 variables.

It may often be advantageous to express a single 0–1 constraint as a collection of 0–1 constraints. We have already seen this in Section 10.1, where constraint (11) was re-expressed as the constraints (14). Here we present a general procedure for expanding any pure 0–1 constraint into a collection of constraints. Ideally we would like to be able to re-express the constraint by a collection of constraints defining the convex hull of feasible 0–1 solutions. In order to make this clear we present an example.

Example 2

\[3\delta_1 + 3\delta_2 - 2\delta_3 + 2\delta_4 + 2\delta_5 \leq 4.\] (5)

\(\delta_i\) are 0–1 variables.

The convex hull of 0–1 solutions which are feasible according to (5) is given by the constraints

\[\delta_1 + \delta_2 - \delta_3 + \delta_4 \leq 1,\] (6)

\[\delta_1 + \delta_2 - \delta_3 + \delta_5 \leq 1,\] (7)

\[\delta_2 + \delta_3 + \delta_4 + \delta_5 \leq 2,\] (8)

\[2\delta_1 + \delta_2 - \delta_3 + \delta_4 + \delta_5 \leq 2,\] (9)

\[\delta_1 + 2\delta_2 - \delta_3 + \delta_4 + \delta_5 \leq 2,\] (10)

together with the trivial constraints \(\delta_i \geq 0\) and \(\delta_i \leq 1\).

Unfortunately no practical procedure has yet been devised for obtaining constraints defining the convex hull of integer solutions corresponding to a single 0–1 constraint. The faces which form the boundary of the feasible region defined by an LP problem are known as 'facets'. For two-variable problems these facets are one-dimensional lines and can easily be visualized in a diagram such as Figure 10.2. With three-variable problems the faces are two-dimensional planes. For \(n\)-variable problems these facets are \((n - 1)\)-dimensional hyperplanes. Hammer, Johnson, and Peled (1975) give a table of the facets of the convex hulls for all 0–1 constraints involving up to five variables. In order to use this table it is first necessary to simplify the constraint to another single constraint using the procedure mentioned above. It is also possible to obtain the convex hull for particular types of constraint involving more than five variables. The expansion of constraint (11) into constraints (14) in Section 10.1 is obviously an instance of this. Although a general procedure for producing the convex hull constraints for a single 0–1 constraint does not yet exist, Balas (1975) gives a procedure for producing some of the facets of the convex hull. Hammer, Johnson, and Peled (1975) and Wolsey (1975) also give similar procedures. They are able to obtain those facets represented by constraints containing only coefficients 0 and \(\pm 1\). For example the procedure would obtain the constraints (6), (7), and (8) of Example 2 above.

We now describe this procedure of Balas. Consider the following pure 0–1 constraint:

\[a_1\delta_1 + a_2\delta_2 + a_3\delta_3 + \cdots + a_n\delta_n \leq a_0.\] (11)

As before there is no loss of generality in considering a '\(\leq\)' constraint with all coefficients non-negative.

Definition

A subset \([i_1, i_2, \ldots, i_r]\) of the indices 1, 2, \ldots, \(n\) of the coefficients in (11) will be called a cover if \(a_{i_1} + a_{i_2} + \cdots + a_{i_r} > a_0\).

Clearly it is not possible for all the \(\delta_i\) corresponding to indices \(i\) within a cover, to be 1 simultaneously. This condition may be expressed by the constraint

\[\delta_{i_1} + \delta_{i_2} + \cdots + \delta_{i_r} \leq n - 1.\] (12)

Definition

A cover \([i_1, i_2, \ldots, i_r]\) is said to be a minimal cover if no proper subset of \([i_1, i_2, \ldots, i_r]\) is also a cover.

Definition

A minimal cover \([i_1, i_2, \ldots, i_r]\) can be extended in the following way:

(i) Choose the largest coefficient \(a_i\) where \(i\) is a member of the minimal cover.

(ii) Take the set of indices \([i_{r+1}, i_{r+2}, \ldots, i_{r+s}]\) not in the minimal cover corresponding to coefficients \(a_i, \ldots, a_{i_{r+s}}\), such that \(a_{i_{r+s}} \geq a_i\).

(iii) Add these set of indices to the minimal cover giving \([i_1, i_2, \ldots, i_{r+s}]\). This new cover is known as an extended cover.
If \([i_1, i_2, \ldots, i_r]\) is a minimal cover it can be augmented to give an extended cover \(i_1, i_2, \ldots, i_{r+1}\). The constraint (12) corresponding to the minimal cover can correspondingly be extended to
\[\delta_{i_1} + \delta_{i_2} + \cdots + \delta_{i_r} \leq r - 1.\]  
(13)

**Definition**

If a minimal cover gives rise to an extended cover which is not a proper subset of any other extended cover arising from a minimal cover with the same number of indices, the original minimal cover is known as a *strong cover*.

Balas shows that if an extended cover arises from a strong cover then the constraints such as (13) encompass all the facets of the convex hull of feasible 0–1 points of (11) which have coefficients 0 or 1.

It is obviously fairly easy to devise a systematic and computationally quick method of generating all strong covers corresponding to a 0–1 constraint and therefore obtaining facet constraints such as (13). In this way any facet of the convex hull of feasible 0–1 points of a constraint can be defined when the facet is represented by a constraint involving only coefficients 0 and 1. Two examples are presented in order to make the process clear.

**Example 3**

This is the same as Example 2, only here we show how the constraints (6), (7), and (8) involving only coefficients 0 and 1 are obtained. It is more convenient to write constraint (5) with positive coefficients as
\[3\delta_1 + 3\delta_2 + 2\delta_3 + 2\delta_4 + 2\delta_5 \leq 6,\]  
(14)
where \(\delta_4 = 1 - \delta_3\). The minimal covers of constraint (14) are
\[\begin{align*}
\{1, & 2, 3\}, & \{1, & 2, 4\}, & \{1, & 2, 5\}, & \{1, & 3, 4\}, & \{1, & 3, 5\}, & \{1, & 4, 5\}, & \{2, & 3, 4\}, & \{2, & 3, 5\}, \{2, & 4, 5\},
\end{align*}\]  
(15) (16) (17) (18) (19) (20) (21) (22) (23)

These minimal covers can be augmented to produce the extended covers
\[\begin{align*}
\{1, & 2, 3\} & \text{from (15)}, & \{1, & 2, 4\} & \text{from (16)}, & \{1, & 2, 5\} & \text{from (17)}, & \{1, & 2, 3, 4\} & \text{from (18) and (21)}, & \{1, & 2, 3, 5\} & \text{from (19) and (22)}, & \{1, & 2, 4, 5\} & \text{from (20) and (23)},
\end{align*}\]  

Since (24), (25), and (26) are proper subsets of (27), (29), and (28) respectively, the first three minimal covers (15), (16), and (17) are not strong covers. The remaining minimal covers (18) to (23) are, however, strong covers and their extensions (27), (28), and (29) give rise to the following facet constraints:
\[\begin{align*}
\delta_1 + \delta_2 + \delta_3 + \delta_4 & \leq 2, & \delta_1 + \delta_2 + \delta_3 + \delta_5 & \leq 2, & \delta_1 + \delta_2 + \delta_4 + 2\delta_5 & \leq 2.
\end{align*}\]  
(30) (31) (32)

Replacing \(\delta_3\) by \(1 - \delta_3\) gives the three facet constraints (6), (7), and (8) in Example 2.

**Example 4**

\[\delta_{n+1} = \begin{cases} 
1 & \text{if a depot is built,} \\
0 & \text{otherwise,}
\end{cases}\]  
\[\delta_i = \begin{cases} 
1 & \text{if customer } i \text{ is supplied from the depot,} \\
0 & \text{otherwise,}
\end{cases}\]  
\[i = 1, 2, \ldots, n.\]

At most \(r (r < n)\) customers may be supplied from the depot if it is built. If the depot is not built, clearly nobody can be supplied from it.

The conditions above may be expressed by the constraint
\[\delta_1 + \delta_2 + \cdots + \delta_n - r\delta_{n+1} \leq 0.\]  
(33)

This is obviously a generalization of the constraint (16) of Section 10.1. It is convenient to express (33) with positive coefficients as
\[\delta_1 + \delta_2 + \cdots + \delta_n + r\delta_{n+1} \leq r,\]  
(34)

where
\[\delta_{n+1} = 1 - \delta_{n+1}.\]  
(35)

The minimal covers of (34) are
\[\{i, n+1\}; \quad i = 1, 2, \ldots, n,\]  
(36)
and all subsets of \([1, 2, \ldots, n]\) such as
\[
\{i_1, i_2, \ldots, i_{r+1}\}
\] (37)
containing \(r + 1\) indices.
Covers (36) cannot be further extended.
All the covers (37) do extend to the same extended cover:
\[
\{1, 2, \ldots, n, n + 1\}.
\] (38)
The minimal covers (36) and (37) in general are of a different size (if \(r \neq 1\)).
Therefore (36) and (38) are all extensions of strong covers and give rise (after substituting \(1 - \delta_{n+1}\) for \(\delta_{n+1}\)) to the constraints:
\[
\delta_i - \delta_{n+1} \leq 0, \quad i = 1, 2, \ldots, n,
\] (39)
and
\[
\delta_1 + \delta_2 + \cdots + \delta_n - \delta_{n+1} \leq r - 1.
\] (40)
These constraints do not necessarily all represent facets, but they are particularly restrictive constraints on the corresponding LP problem and include all the facets with coefficients 0 or 1. It would therefore be advantageous to append constraints (39) and (40) to the original constraint (33) in a model.

This way of obtaining particularly 'strong' extra constraints to add to an IP model could prove of value in the MARKET SHARING problem of Part 2.

Simplifying Collections of Constraints

So far we have described ways of simplifying individual constraints involving 0–1 variables. What we would ideally like to do would be to simplify the collection of all the constraints into a collection of constraints defining the convex hull of feasible integer points. It should be pointed out that simplifying constraints individually will not generally suffice although it may be computationally helpful towards the ultimate aim of solving the IP problem more easily. This is demonstrated by an example.

Example 5

Maximize \(\delta_1 + 2\delta_2 + \delta_3\)
subject to \(2\delta_1 + 3\delta_2 + 2\delta_3 \leq 3,\) \((41)\)
\(\delta_1 + \delta_2 - 2\delta_3 \leq 0.\) \((42)\)
\(\delta_1, \delta_2, \delta_3\) are 0–1 variables.
The optimal solution to the associated LP problem is
\(\delta_1 = 0, \quad \delta_2 = \frac{1}{2}, \quad \delta_3 = \frac{1}{2},\) giving an objective of \(\frac{1}{2}\).

If constraint (41) is replaced by the constraints representing its facets we obtain constraint (43) in the model below. Constraints (44) and (45) come from the two facets of (42). The simplified model is then:

Maximize \(\delta_1 + 2\delta_2 + \delta_3\)
subject to \(\delta_1 + \delta_2 + \delta_3 \leq 1,\) \((43)\)
\(\delta_1 - \delta_3 \leq 0,\) \((44)\)
\(\delta_2 - \delta_3 \leq 0.\) \((45)\)
The optimal solution to the associated LP problem for this reformulated model is
\(\delta_1 = 0, \quad \delta_2 = \frac{1}{2}, \quad \delta_3 = \frac{1}{2},\) giving an objective of \(\frac{1}{2}\).

Clearly simplifying constraints individually has not constrained the whole model sufficiently to guarantee an integer solution to the IP model although the objective value is closer to the integer optimum:
\(\delta_1 = 0, \quad \delta_2 = 0, \quad \delta_3 = 1,\) giving an objective of 1.

Unfortunately no practical procedure is known for producing the convex hull of the feasible 0–1 points corresponding to a general collection of pure 0–1 constraints. In fact if such a computationally efficient procedure were known we could reduce all PIP problems to LP problems and dispense with PIP. Procedures of this sort do exist for special restricted classes of PIP problem. The best known is for the matching problem which was mentioned in Section 9.5. The main reference to the problem is Edmonds (1965).

Partial results exist enabling one to obtain some of the facets of the convex hull for some types of PIP problem. Hammer, Johnson, and Peled (1975) have a procedure for generating the facet constraints involving only coefficients 0 and 1 for a class of PIP problems they call 'regular'. Within this class of problems are the set covering problem and the knapsack problem. Clearly by being able to cope with the knapsack problem they can also obtain the facets obtained by Balas for a single constraint. Apart from what has already been described none of these partial results seems as yet to give a valuable formulation tool for practical problems and they will not therefore be described further.

It should also be pointed out that procedures have been devised for doing the reverse of what we have described here. A collection of pure integer equality constraints can be progressively combined with one another to obtain a single equality constraint giving a knapsack problem. Bradley (1971) describes such a procedure. Unfortunately the resulting coefficients are often enormous. There is little interest in such a procedure if the corresponding LP problem is to be used as a starting point for solving the IP problem. The general effect of aggregating constraints in this way will be to weaken rather
than restrict the corresponding LP problem. Chvatal and Hammer (1975) also
describe a procedure for combining pure 0–1 inequality constraints.

Most of the discussion in this section has concerned adding further
restrictions to an IP model in order to restrict the corresponding LP model.
Such extra restrictions can be viewed in the context of cutting planes
algorithms for IP. We are adding cuts to a model in order to eliminate some
possible fractional solutions. Most algorithms which make use of cutting
planes generate the extra constraints in the course of optimization. Our
interest here, in a book on model building, is only in adding cuts to the initial
model.

Discontinuous Variables

It is sometimes necessary to restrict a continuous variable to segments of
continuous values, e.g.

\[ x = 0 \quad \text{or} \quad a \leq x \leq b \quad \text{or} \quad x = c, \quad (46) \]

where \(0 < a < b < c\).

A straightforward approach to follow is that described for disjunctive
constraints in Section 9.4, where 0–1 variables are used to indicate each of the
three (or more) possibilities. A constraint of type \(3\) in that section the forces
the condition.

There is an alternative formulation which has been suggested by Brearley
(1975), following a more conventional formulation of a blending problem with
logical restrictions by Thomas, Jennings, and Abbott (1978). This is

\[ x = ay_1 + by_2 + c\delta_2, \quad (47) \]

\[ \delta_1 + y_1 + y_2 = 1, \quad (48) \]

where \(\delta_1\) and \(\delta_2\) are 0–1 integer variables and \(y_1\) and \(y_2\) are (non-negative)
continuous variables.

Condition (46) can clearly be generalized or specialized. A common special
case is that of a semicontinuous variable, i.e.

\[ x = 0 \quad \text{or} \quad x \geq a \quad (a > 0). \quad (49) \]

In order to model this an upper bound \((M)\) must be specified for \(x\), giving the
formulation

\[ x = ay_1 + My_2, \quad (50) \]

\[ \delta + y_1 + y_2 = 1, \quad (51) \]

where \(\delta\) is a 0–1 variable and \(y_1\) and \(y_2\) are (non-negative) continuous
variables.

An Alternative Formulation for Disjunctive Constraints

An alternative formulation for a disjunction of constraints has been given by

Jeroslow and Lowe (1984). In addition they construct a 'Theory of Mixed
Integer Programming Representability' which begins to put the whole subject
on systematic foundations. The use of Jeroslow's 'disjunctive formulations'
has a considerable theoretical advantage as well as manifesting a practical
advantage in large models. Computational experience is reported in Jeroslow

A comprehensive discussion of the subject is given in the lecture notes of

Suppose we have a disjunction of constraints such as (1) in Section 9.4 where
each \(R_k\) represents a set of constraints:

\[ \sum_j a_{ijk}x_j \leq b_{ik}, \quad i = 1, 2, \ldots, m_k \quad (52) \]

We will suppose that each set of constraints \((52)\) in this disjunction has a
closed feasible region. If necessary this may be achieved by using known
bounds on the quantities in the constraints as is also required in the
conventional formulation. In fact even if the feasible region is not closed it is
not always necessary to use such bounds in this new mode of formulation. The

Each variable \(x_j\) is split into separate variables \(x_{jk}\) with constraints:

\[ x_j = x_{j1} + x_{j2} + \ldots + x_{jN} \quad (53) \]

The new variables replace the original variables in the set of constraints \(R_k\)
to which they correspond giving constraints:

\[ \sum_j a_{ijk}x_{jk} - b_{ik}\delta_k \leq 0, \quad i = 1, 2, \ldots, m_k \quad (54) \]

\[ \sum_k \delta_k = 1 \quad (55) \]

where \(\delta_k\) are 0–1 integer variables.

Constraint (55) forces exactly one \(\delta_k\) to be 1 and the others to be zero. If \(\delta_k\) is
0 then constraints \((54)\) (having a closed feasible region) force the correspond-
ing \(x_{jk}\) all to be zero. Hence for each \(j\) only one component \(x_{jk}\) can be
non-zero making it, by constraint \((53)\), equal to \(x_j\) and so guaranteeing the
constraints corresponding to \(R_k\).

It can be shown that if each set of constraints \((52)\) is a sharp formulation
of \(R_k\) then this resultant formulation of the disjunction is also sharp, i.e.
the linear programming relaxation gives the convex hull of feasible integer
solutions. If, for example, the variables in \((52)\) are all continuous variables
then the property holds.

In section 9.2 it was shown that logical conditions often can be specified in
more than one way. A well-known result in Boolean Algebra is that any
proposition can be expressed in a standard form known as Disjunctive Normal
Form which uses only the and (.), or (\lor) and not (\neg) connectives e.g.

\[(R_1 \land R_2 \land \ldots \land R_{m_1}) \lor (R_2 \land R_2 \land \ldots \land R_{m_2}) \lor \ldots \lor (R_{N_1} \land R_{N_2} \land \ldots \land R_{N_{L_{max}}})\]  \hspace{1cm} (56)

where there is a disjunction of clauses each of which is made up of a conjunction of statements \(R_{ij}\) (some of which may be negated statements). Jeroslows suggests that it is generally better to express a model using this disjunctive normal form and then use his disjunctive formulation. It is theoretically possible to arrive at a sharp formulation for any IP this way taking account of the fact that any (bounded) integer variable represents a disjunction of possibilities e.g.

\[x = 0 \lor x = 1 \lor x = 2 \lor \ldots \lor x = m.\]  \hspace{1cm} (57)

In practice the number of variables created by such a formulation can be prohibitively large since the disjunctive formulation splits variables into components in the manner described above. A compromise must generally be adopted which will not be sharp but is often tighter than a conventional formulation. In practice many simplifications of the resultant formulation are often possible (using for example a reduction procedure).

Two other observations of Jeroslows are worth making here. He points out that it is desirable, from the sharpness point of view, to apply the and connective before specifying an IP formulation. This is in contrast to creating IP formulations of components of a problem and then applying the and connective (i.e. putting all the resulting constraints together). This is because (in set notation)

\[\text{Con}(S \cap T) \subseteq \text{Con}(S) \cap \text{Con}(T)\]  \hspace{1cm} (58)

where \(S\) and \(T\) are sets and \text{Con} is the operation of taking the convex hull.

It is also desirable to aim for formulations which are hereditarily sharp, i.e. when certain integer variables are fixed (by the branch and bound algorithm) the resulting submodels remain sharp.

10.3 Economic Information Obtainable by Integer Programming

We saw in Section 6.2 that in addition to the optimal solution values of an LP problem important additional economic information can also be obtained from such quantities as the shadow prices and reduced costs. The dual LP model was also shown to have an important economic interpretation in many situations. In addition a close relationship between the solution to the original model and its dual exist.

It is worth just briefly pointing out how the duality relationship fails in IP. Suppose we have an IP maximization problem \(P\). Corresponding to \(P\) we have the LP problem \(P'\). As long as \(P'\) is feasible and not unbounded we have a solvable dual problem \(Q'\). From duality in LP we know that

maximum objective of \(P'\) = minimum objective of \(Q'\).

By imposing extra integrality requirements on \(P'\) we obtain the IP problem \(P\). Clearly since \(P\) is more constrained than \(P'\) we have

maximum objective of \(P\) ≤ maximum objective of \(P'\) = minimum objective of \(Q'\).

The minimum objective of \(Q'\) is the smallest upper bound we can obtain for the objective of \(P\) by a set of valuations on the constraints of \(P\). This contrasts with the LP case where the dual values provide a strict upper bound (i.e. a bound that is obtained by the optimum) for the objective. In consequence the difference between the maximum objective value of \(P\) and the maximum objective of \(P'\) (or minimum objective of \(Q'\)) is sometimes known as a duality gap. It can be regarded (rather loosely) as a measure of how inadequate any dual values will be when used as shadow prices.

In this section we attempt to obtain corresponding economic information from an IP model to that obtainable from an LP model. It will be seen that this information is much more difficult to come by in the case of IP and is, in some cases, rather ambiguous. In order to demonstrate the difficulties we will consider a 'product mix' problem where the variables in the model represent quantities of different products to be made and the constraints represent limitations on productive capacity. It is only meaningful to make integral numbers of each product.

Example 1

Maximize \[12\gamma_1 + 5\gamma_2 + 15\gamma_3 + 10\gamma_4\]

subject to \[5\gamma_1 + 2\gamma_2 + 9\gamma_3 + 12\gamma_4 \leq 15,\] \hspace{1cm} (1)

\[2\gamma_1 + 3\gamma_2 + 4\gamma_3 + \gamma_4 \leq 10,\] \hspace{1cm} (2)

\[3\gamma_1 + 2\gamma_2 + 4\gamma_3 + 10\gamma_4 \leq 8,\] \hspace{1cm} (3)

\[\gamma_1, \gamma_2, \gamma_3, \gamma_4 \geq 0.\]

The \(\gamma_i\) are general integer variables.

The optimal integer solution is \(\gamma_1 = 2, \gamma_2 = 1, \gamma_3 = 0,\) and \(\gamma_4 = 0,\) giving an objective value of 29.

For comparison the optimal solution to the corresponding LP problem is \(\gamma = 2, \gamma_2 = 0, \gamma_3 = 0,\) and \(\gamma_4 = 0,\) giving an objective value of 32.

In addition to the fractional solution of the LP problem we would be able to obtain answers to such questions as the following:

(Q1) What is the marginal value of increasing fully utilized capacities?

(Q2) How much should a non-manufactured product be increased in price to make it worth manufacturing?

We saw in Section 6.2 that the answers to Q1 came from the shadow prices on
the corresponding constraints. These shadow prices represented valuations which could be placed on the capacities. Once these optimal valuations have been obtained the optimal manufacturing policy can be deduced by simple accounting. Moreover, the total valuation for all the capacities implied by these shadow prices is the same as the profit obtainable by the optimal manufacturing policy.

Unfortunately there are no such neat values which may be placed on capacities in the case of an IP model. For example, the capacity represented by constraint (1) is not fully used up in the IP optimal solution. In an LP problem if a constraint has slack capacity, as in this case, it represents a free good as described in Section 6.2 and has a zero shadow price. Such a constraint could be omitted from the LP model and the optimal solution would be unchanged. In this case constraint (1) cannot be omitted without changing the optimal solution. We would therefore like to give the constraint some economic valuation. This gives the first important difference that must exist between any valuations of constraints in IP and shadow prices in LP:

(A) If a constraint has positive slack it does not necessarily represent a free good and may therefore have a positive economic value.

Why this should be so is fairly easy to see since, although there is no virtue in slightly increasing the right-hand side value of 15 in constraint (1), there clearly is virtue in increasing it by at least 3 since we could then bring two of $\gamma_3$ into the solution in place of two of $\gamma_1$ and one of $\gamma_2$.

Even if we admit positive valuations on unsatisfied as well as satisfied capacities it may still be impossible to arrive at a method of decision making through pricing in a similar way to that described in Section 6.2 for LP. This is demonstrated by the following example.

Example 2

Maximize $4\gamma_1 + 3\gamma_2 + \gamma_3$

subject to $2\gamma_1 + 2\gamma_2 + \gamma_3 \leq 7,$

where $\gamma_1, \gamma_2, \gamma_3 \geq 0$ and take integer values.

The optimal solution is $\gamma_1 = 3, \gamma_2 = 0, \gamma_3 = 1$. We will attempt to find a valuation for the constraint which will produce this answer.

Suppose we give the constraint a 'shadow price' (or accounting value) of $\pi$, then to make $\gamma_1$ profitable we must have $\pi \leq 1$. This, however, implies $2\pi < 3$ making $\gamma_2$ also profitable. We know from the optimal solution that $\gamma_3$ is worth making but that $\gamma_2$ is not.

This example demonstrates a second difference between any economic valuation of constraints in IP in comparison with shadow prices in LP:

(B) For general IP problems no valuations will necessarily exist for the constraints which allow the optimal solution to be obtained in a similar manner to the LP case.

By 'constraints' in (B) we must as usual exclude the feasibility constraints $\gamma_j \geq 0$.

One way out of the dilemma posed by the IP model above is to obtain constraints representing the convex hull of feasible integer solutions. (For MIP models we would of course consider the convex hull of points with integer coordinates in the dimensions representing integer variables as mentioned in Section 10.1.) The model can then be treated as an LP problem and shadow prices obtained with desirable properties. Although a procedure for obtaining the convex hull of integer solutions is computationally often impractical, as discussed in the last section, it can be applied to certain models making useful economic information possible. This is demonstrated on a particular type of problem, the shared fixed cost problem, in Example 4 below. In spite of the general computational difficulties it is still worth considering this as a theoretical solution to our dilemma. For the purposes of explanation we have reformulated the IP model in Example 1 above, using constraints for the convex hull of feasible integer points. This gives the model below.

Example 3

Maximize $12\gamma_1 + 5\gamma_2 + 15\gamma_3$

subject to $\gamma_1 + \gamma_2 + \gamma_3 \leq 2,$

$\gamma_1 + \gamma_2 + \gamma_3 \leq 3,$

$2\gamma_1 + \gamma_2 + 3\gamma_3 \leq 5,$

$\gamma_3 \leq 1,$

where $\gamma_1, \gamma_2, \gamma_3 \geq 0$ and take integer values. (When integer, $\gamma_4$ is clearly forced to be zero by constraint (3).)

The shadow prices on the new constraints are

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Shadow Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5)</td>
<td>2</td>
</tr>
<tr>
<td>(6)</td>
<td>0</td>
</tr>
<tr>
<td>(7)</td>
<td>5</td>
</tr>
<tr>
<td>(8)</td>
<td>0</td>
</tr>
</tbody>
</table>

N.B. Since Example 3 is degenerate there are alternative shadow prices, as explained in Section 6.2.

Unfortunately it is not clear how the new constraints (5), (6), (7), and (8) defining the convex hull of Example 1 can be related back to the original constraints (1), (2), and (3). It is therefore difficult to apply the shadow prices above to give meaningful valuations for the physical constraints of the original
model. An attempt has been made to do this by Gomory and Baumol (1960). They apply a cutting planes algorithm to the IP model, successively adding constraints until an integer solution is obtained. Then shadow prices are obtained for the original constraints and for the added constraints. The shadow prices for the added constraints are imputed back to the original constraints from which they were derived. Unfortunately the valuations they obtain for the original constraints are not unique and depend on the way in which the cutting planes algorithm is applied. Also they have to include the feasibility conditions \( y_i \geq 0 \) among their original constraints. As a result these constraints may end up being given non-zero economic valuations. In LP such valuations could be regarded as the reduced costs on the variables \( y_i \) and no difficulty would arise. Variables with positive reduced costs would be out of the optimal solution. With IP using this procedure it would be perfectly possible for the feasibility condition \( y_i \geq 0 \) to have non-zero economic values (suggesting that in some sense \( y_i \geq 0 \) was a 'binding' constraint) and for \( y_i \) to be in the optimal solution at a positive level. One way of justifying this would be to regard the economic valuation given to the feasibility constraint as a cost associated with the indivisibility of \( y_i \). Not only should \( y_i \) be charged according to the use it makes of scarce capacities, it should also be charged an extra amount in view of the fact it can only come in integral quantities.

The fact that a constraint such as \( y_i \geq 0 \) may have a non-zero valuation in the Gomory–Baumol system yet \( y_i \) may not be at zero level is a special case of a more general difference between the Gomory–Baumol prices in IP and the shadow prices in LP:

(C) A free good as represented by a constraint in IP does not necessarily have a zero Gomory–Baumol price attached to it.

This difference is not as serious as it might first seem since a similar situation can happen in LP. We saw in Section 6.2 that with degeneracy there are alternate dual solutions, some of which may give non-zero shadow prices to (alternatively) redundant constraints. This also indicates that the problem on non-uniqueness in the Gomory–Baumol prices is not confined to IP. It clearly also happens, although to a much less serious extent, with degenerate problems in LP.

The exact way of obtaining limited economic information from a MIP model which is used quite widely in practice should be mentioned. This is simply to take this information from the LP subproblem at the node in the solution tree which gave the optimal integer solution. Such information may well be unreliable since the integer variables should only change by discrete amounts while the economic information results from the effect of marginal changes. Other integer variables (particularly 0–1 variables) will have become fixed by the bounds imposed in the course of evaluating the solution tree and it will not be possible to evaluate the effect of any changes on these variables.

A variation of this way of obtaining economic information from a MIP model is to 'fix' all integer variables at their optimal values and only consider the effect of marginal changes on the continuous variables. This procedure has something to recommend it since the integer variables usually represent major operating decisions. Given that these decisions have been accepted, the economic effects of marginal changes within the basic operating pattern may be of interest.

An example of the need to value the constraints of a MIP model is provided by the TARIFF RATES (POWER GENERATION) problem in Part 2. The rates at which electricity is sold on different tariffs implicitly value the constraints. Different ways of doing this are discussed with the solution of the model in Part 4.

In spite of the difficulties in getting meaningful subsidiary economic information out of an IP model there are circumstances where useful information can be obtained by reformulation of the model. This is discussed by Williams (1981). We will illustrate this in the example below by reformulating a model using the ideas contained in Section 10.2. The problem considered involves shared fixed costs and is described by Rhys (1970), to whom these ideas are due.

**Example 4**

In the network of Figure 10.3 the nodes represent capital investments with the costs associated with them. The arcs represent money-making activities with the estimated revenues associated with them. To carry out any activity (arc) it is necessary to use both the resources represented by the nodes at either end of the arc. The problem is to share the capital investment (fixed) cost associated with a node in some optimal way among the activities associated with the arcs joining the node, e.g. how should the capital cost of node D be shared among the arcs AD, BD, CD, and DE? An illustrative way of viewing this problem is to think of the nodes as stations with the arcs as railways between those stations.

In order to show how the required economic information can arise through reformulating an IP model we will first consider a rather different problem.

Suppose we wish to decide which nodes (stations) to cut in order to make the
whole network (railway system) as profitable as possible. It must be borne in mind that cutting out a node (station) necessitates cutting out all the arcs (lines) leading into it. This problem can easily be formulated as an IP problem using the following 0–1 variables:

\[
\delta_i = \begin{cases} 
1 & \text{if node } i \text{ is kept,} \\
0 & \text{if node } i \text{ is cut out;}
\end{cases}
\]

\[
\delta_{ij} = \begin{cases} 
1 & \text{if arc } (ij) \text{ is kept,} \\
0 & \text{if arc } (ij) \text{ is cut;}
\end{cases}
\]

where \(i, j\), are A, B, C, D, and E.

The objective to be maximized is

\[
-5\delta_A - 4\delta_B - 8\delta_C - 8\delta_D - 4\delta_E + 7\delta_{AB} + 3\delta_{AC} + 3\delta_{AD} + 8\delta_{BD} + 4\delta_{CD} + \delta_{DE} + 2\delta_{CE}. 
\]  

(9)

The conditions to be modelled are that certain acts require certain nodes, i.e.

\[
\delta_{ij} = 1 \rightarrow \delta_i = 1, \ \delta_j = 1. 
\]

(10)

We saw in Section 10.1 that such a condition may be modelled two ways using either one or two constraints as

\[
-\delta_i - \delta_j + 2\delta_{ij} \leq 0
\]

(11)

or

\[
-\delta_i + \delta_{ij} \leq 0, \\
-\delta_j + \delta_{ij} \leq 0.
\]

(12)

The second formulation has the advantage that the model will be totally unimodular and can be solved as an LP problem yielding an integer optimal solution. Geometrically we have specified the convex hull of feasible integer points by the constraints (12). Since we now have an LP problem we obtain well defined shadow prices on the constraints. In this example the shadow prices have the following interpretation.

The shadow price on \(-\delta_i + \delta_{ij} \leq 0\) is the amount of the capital cost of node \(i\) which should be met by revenue from arc \((ij)\).

Similarly the shadow price on \(-\delta_j + \delta_{ij} \leq 0\) is the amount of the capital cost of node \(j\) which should be met by revenue from arc \((ij)\).

We have clearly found a way of sharing the capital costs of the nodes among the arcs. Should any activity not be able to meet the capital cost demanded of it, it should be cut out. This allocation of capital costs will be such as to lead to the most profitable network. Using the numbers given on the network in Figure 10.3 the following shadow prices result as shown in Table 10.1.

An interesting observation following from duality in LP is that dividing the capital costs of the nodes up in other ways among the arcs could not lead to a more profitable network and could well lead to a less profitable one.

It should have become apparent from all the discussion in this section that there is no generally satisfactory way of getting the subsidiary economic information from an IP model that often proves so valuable in the case of LP. This topic represents a considerable gap in mathematical programming theory. The subject is more fully discussed in Williams (1979).

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Shadow price</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-\delta_A + \delta_{AB} \leq 0)</td>
<td>3</td>
</tr>
<tr>
<td>(-\delta_A + \delta_{AC} \leq 0)</td>
<td>3</td>
</tr>
<tr>
<td>(-\delta_A + \delta_{AD} \leq 0)</td>
<td>3</td>
</tr>
<tr>
<td>(-\delta_A + \delta_{BD} \leq 0)</td>
<td>3</td>
</tr>
<tr>
<td>(-\delta_B + \delta_{AB} \leq 0)</td>
<td>1</td>
</tr>
<tr>
<td>(-\delta_B + \delta_{BC} \leq 0)</td>
<td>5</td>
</tr>
<tr>
<td>(-\delta_B + \delta_{BD} \leq 0)</td>
<td>0</td>
</tr>
<tr>
<td>(-\delta_C + \delta_{AC} \leq 0)</td>
<td>0</td>
</tr>
<tr>
<td>(-\delta_C + \delta_{BC} \leq 0)</td>
<td>3</td>
</tr>
<tr>
<td>(-\delta_D + \delta_{AD} \leq 0)</td>
<td>1</td>
</tr>
<tr>
<td>(-\delta_D + \delta_{BD} \leq 0)</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 10.4

10.4 Sensitivity Analysis and the Stability of a Model

We saw in Section 6.3 that having built and solved an LP model it was very important to see how sensitive the answer was to changes in the data from which the model was constructed. Using ranging procedures on the objective and right-hand side coefficients some insight into this could be obtained. In addition it was shown how a model could, to some extent, be built in order to behave in a 'stable' fashion. Our considerations here are exactly the same as those in Section 6.3 only they concern IP models. As in the last section we will see that IP models present considerable extra difficulties. This section falls into two parts. Firstly we will consider ways of testing the sensitivity of the solution
of an IP model. Secondly we will consider how models may be built in order that they may exhibit stability.

Sensitivity Analysis and Integer Programming

A theoretical way of doing sensitivity analysis on the objective coefficients of a model would be to replace the constraints by constraints representing the convex hull of feasible integer points. The model could then be treated as an LP model and objective ranging performed as described in Section 6.3.

For those PIP models where a reformulation easily yields the constraints for the convex hull this is fairly straightforward. Otherwise it is not a practical way of approaching the problem. Nor does it give a way of performing right-hand side ranging.

For MIP models solved by the branch and bound method a sensitivity analysis can be performed on the LP subproblem at the node giving the optimal integer solution. Alternatively the integer variables can be fixed at their optimal values and a sensitivity analysis performed on the continuous part of the problem. These approaches clearly have the same drawbacks as those apparent when using similar approaches to derive economic information from an IP model as described in Section 10.3.

The only really satisfactory method of sensitivity analysis in IP involves solving the model again with changed coefficients and comparing optimal solutions. Obviously the subsequent time to solve the model should be able to be reduced by exploiting the knowledge of the previous solution.

Building a Stable Model

In an LP model the optimal value of the objective function varies continuously as the right-hand side and objective coefficients are changed. In an IP model this may not happen. We consider the following very simple example.

Example 1

Maximize $40\delta_1 + 35\delta_2 + 15\delta_3 + 8\delta_4 + 9\delta_5$
subject to $8\delta_1 + 8\delta_2 + 5\delta_3 + 4\delta_4 + 3\delta_5 \leq 16$.

$\delta_1, \delta_2, \delta_3, \delta_4, \delta_5$ are 0-1 variables.

The optimal solution to this model is $\delta_1 = 1$ and $\delta_2 = 1$ giving an objective value of 75.

If, however, the right-hand side value of 16 is reduced by a small amount the optimal solution changes to $\delta_1 = 1, \delta_4 = 1, \delta_5 = 1$ giving an objective value of 57.

The optimal value of the objective function is obviously not a continuous function of the right-hand side coefficient. In many practical situations this is unrealistic. Suppose constraint (1) represented a budgetary limitation and the 0-1 variables referred to capital investments. It is very unlikely that a small decrease in the budget would cause us radically to alter our plans. A more likely occurrence is that the budget would be stretched slightly at some increased cost or the cost of one of the capital investments would be trimmed slightly. It is important that if this is the case this should be represented in the model. As it stands the above example represents a poor model of the situation.

One way to remodel constraint (1) is to use the device described in Section 3.3 in order to allow constraints to be violated at a certain cost. A surplus (continuous) variable $u$ is added into constraint (1) and given a cost (say 20) in the objective. This results in the model:

Maximize $40\delta_1 + 35\delta_2 + 15\delta_3 + 8\delta_4 + 9\delta_5 - 20u$
subject to $8\delta_1 + 8\delta_2 + 5\delta_3 + 4\delta_4 + 3\delta_5 - u \leq 16$.

For a right-hand side of 16 the optimal solution is still $\delta_1 = \delta_2 = 1$, giving an objective value of 75.

If the right-hand side value of 16 is reduced slightly the effect of $u$ will be to 'top the budget up' to 16 at a unit cost of 20. For example, if the right-hand side falls to 15, $u$ will become 1. We will retain the same optimal solution but the objective will fall by 10 to 65. As the right-hand side is further reduced the optimal value of the objective will continue gradually to fall until we reach a right-hand side value of 15. We will then have the solution $\delta_1 = 1, \delta_4 = 1, \delta_5 = 1$ as an alternative optimum. If the right-hand side is further reduced we will transfer to this alternative optimum. It can be seen that this device of adding a surplus variable to the problem with a certain cost has two desirable effects on the model:

(i) The optimal objective value becomes a continuous function of the right-hand side coefficient.

(ii) The optimal solution values do not change 'suddenly' as the right-hand side coefficient changes. They are said to be 'semi-continuous' functions of the right-hand side.

In some applications we might wish to add a slack (continuous) variable $u$ as well; giving $u$ a cost in the objective (of say 8). This would result in the following model:

Maximize $40\delta_1 + 35\delta_2 + 15\delta_3 + 8\delta_4 + 9\delta_5 - 20u - 8u$
subject to $8\delta_1 + 8\delta_2 + 5\delta_3 + 4\delta_4 + 3\delta_5 - u + u = 16$.

This topic will not be pursued further here since it has been fully covered for LP problems in Section 6.3. It is important to notice, however, the desirability of forcing the optimal solution of an IP model to very 'continuously' with the data coefficients. Clearly for many logical type constraints which appear in MIP models it is meaningless to use a device such as that above. There are
some classes of MIP model where the continuity property can be shown to hold without further reformulation. This subject is discussed more deeply by A. C. Williams (1989).

10.5 When and How to Use Integer Programming

In this section we try, very briefly, to summarize some of the points made in the preceding three chapters as a quick guide to using IP.

(1) If a practical problem has any of the characteristics described in Section 8.2 it is worth considering the use of an IP model.

(2) Before a decision is made to build an IP model an estimate should be made of the potential size. If the number of integer variables is more than a few hundred then, unless the problem has a special structure, it is probable that IP will be computationally too costly.

(3) A close examination of the structure of the IP model which would result from the problem is always worthwhile. Should the model be a PIP model and have a totally unimodular structure then LP can be used and models involving thousands of constraints and variables solved in a reasonable period of time. If the structure is a PIP model but not totally unimodular it is worth seeing if it can easily be transformed into a known totally unimodular structure, as described in Section 10.2. For MIP models or models where there is no apparent way of producing a unimodular structure it is often possible to constrain the corresponding LP problem more tightly. If it is apparent that the IP model will have one of the other special structures mentioned in Section 9.5, it is worth examining the literature and computational experience with the class of problem to get an impression of the difficulty.

(4) Before embarking on the full scale model it is worth building a model for a much smaller version of the problem. Experiments should be performed on this model to find out how easy it is to solve. If necessary, reformulations such as those described in Section 10.2 should be tried. Different solution strategies as mentioned in Section 8.3 should also be experimented with.

(5) If the problem appears too difficult to solve as an IP model after carrying out the above investigations some heuristic method will have to be used. Much literature exists on different heuristic algorithms for Operational Research problems but this topic is beyond the scope of this book. For an apparently difficult problem where an IP model still seems worthwhile it may also be worth some time being spent on a heuristic approach to get a fairly good, though probably not optimal solution. This good solution can then be exploited in the tree search as a cut-off value for the objective function as described in Section 8.3.

(6) Having built an IP model it is very important to use an intelligent solution strategy, using, if possible, one's knowledge of the practical problem. This has been mentioned briefly in Section 8.3 but is, in the main, beyond the

Finally it should be pointed out that theoretical and computational progress in IP is being made all the time, making it possible to solve larger and more complex models.
CHAPTER 12

The Problems

There is no significance in the order in which the following 20 problems are presented. Some of them are easy to formulate and present no computational difficulties in solution. Others are more difficult in either one of these respects or both. It will be found that some of the problems can be solved with linear programming, others require integer programming or separable programming.

It is suggested that the reader attempts to formulate those problems which interest him before consulting the suggested formulations and solutions in Parts 3 and 4. If he has available a computer he may wish to use this on his model. Alternatively, he may attempt an intuitive (heuristic) approach to some of the problems using original methods of his own. He can compare his answer with the optimal one given in Part 4.

If the reader does wish to follow the recommended course of solving the models using a computer package he is strongly advised to use a matrix generator/language, as discussed in Sections 3.5 and 4.3. This enables him to concentrate on the structure of the model as well as facilitating error detection and greatly reducing data preparation.

In Section 3.5 some discussion has been given of the matrix generator/language MAGIC. Further details of its use can be found in Day (1984). Each of the formulations suggested in Part 3 has been modelled using MAGIC. Should the reader wish to obtain these formulations he should write to the author at the Faculty of Mathematical Studies, The University, Highfield, Southampton SO9 5NH, UK.

12.1 Food Manufacture

A food is manufactured by refining raw oils and blending them together. The raw oils come in two categories:

<table>
<thead>
<tr>
<th>Raw Oils</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vegetable Oils</td>
<td>VEG 1</td>
</tr>
<tr>
<td></td>
<td>VEG 2</td>
</tr>
<tr>
<td>Non-vegetable Oils</td>
<td>OIL 1</td>
</tr>
<tr>
<td></td>
<td>OIL 2</td>
</tr>
<tr>
<td></td>
<td>OIL 3</td>
</tr>
</tbody>
</table>
Each oil may be purchased for immediate delivery (January) or bought on the future's market for delivery in a subsequent month. Prices now, and in the future's market are given below (in £/ton):

<table>
<thead>
<tr>
<th></th>
<th>VEG 1</th>
<th>VEG 2</th>
<th>OIL 1</th>
<th>OIL 2</th>
<th>OIL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>110</td>
<td>115</td>
</tr>
<tr>
<td>February</td>
<td>130</td>
<td>130</td>
<td>110</td>
<td>90</td>
<td>115</td>
</tr>
<tr>
<td>March</td>
<td>110</td>
<td>140</td>
<td>130</td>
<td>100</td>
<td>95</td>
</tr>
<tr>
<td>April</td>
<td>120</td>
<td>110</td>
<td>120</td>
<td>120</td>
<td>125</td>
</tr>
<tr>
<td>May</td>
<td>100</td>
<td>120</td>
<td>150</td>
<td>110</td>
<td>105</td>
</tr>
<tr>
<td>June</td>
<td>90</td>
<td>100</td>
<td>140</td>
<td>80</td>
<td>135</td>
</tr>
</tbody>
</table>

The final product sells at £150 per ton.

Vegetable oils and non-vegetable oils require different production lines for refining. In any month it is not possible to refine more than 200 tons of vegetable oils and more than 250 tons of non-vegetable oils. There is no loss of weight in the refining process and the cost of refining may be ignored.

It is possible to store up to 1000 tons of each raw oil for use later. The cost of storage for vegetable and non-vegetable oil is £5 per ton per month. The final product cannot be stored. Nor can refined oils be stored.

There is a technological restriction of hardness on the final product. In the units in which hardness is measured this must lie between 3 and 6. It is assumed that hardness blends linearly and that the hardnesses of the raw oils are

<table>
<thead>
<tr>
<th></th>
<th>VEG 1</th>
<th>VEG 2</th>
<th>OIL 1</th>
<th>OIL 2</th>
<th>OIL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEG 1</td>
<td>8·8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEG 2</td>
<td>6·1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OIL 1</td>
<td>2·0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OIL 2</td>
<td>4·2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OIL 3</td>
<td>5·0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What buying and manufacturing policy should the company pursue in order to maximize profit?

At present there are 500 tons of each type of raw oil in storage. It is required that these stocks will also exist at the end of June.

Investigate how total monthly profit and buying and manufacturing policy should change for different prices in the future's market. The price changes to be considered are an x% increase in vegetable oils and a 2x% increase in non-vegetable oils in the February market. For March these increases are 2x% and 4x%. These increases continue linearly upwards for the rest of the year.

The policy changes necessary and their effect on total profit should be mapped out for different values of x (up to 20).

12.2 Food Manufacture 2

It is wished to impose the following extra conditions on the food manufacture problem:

1. The food may never be made up of more than three oils in any month.
2. If an oil is used in a month at least 20 tons must be used.
3. If either of VEG 1 or VEG 2 are used in a month then OIL 3 must also be used.

Extend the food manufacture model to encompass these restrictions and find the new optimal solution.

12.3 Factory Planning

An engineering factory makes seven products (PROD 1 to PROD 7) on the following machines: four grinders, two vertical drills, three horizontal drills, one borer, and one planer. Each product yields a certain contribution to profit (defined as £/unit selling price minus cost of raw materials). These quantities (in £/unit) together with the unit production times (hours) required on each process are given below. A dash indicates that a product does not require a process.

<table>
<thead>
<tr>
<th></th>
<th>PROD 1</th>
<th>PROD 2</th>
<th>PROD 3</th>
<th>PROD 4</th>
<th>PROD 5</th>
<th>PROD 6</th>
<th>PROD 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contribution to profit</td>
<td>10</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>11</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Grinding</td>
<td>0·5</td>
<td>0·7</td>
<td>—</td>
<td>—</td>
<td>0·3</td>
<td>0·2</td>
<td>0·5</td>
</tr>
<tr>
<td>Vertical drilling</td>
<td>0·1</td>
<td>0·2</td>
<td>—</td>
<td>—</td>
<td>0·5</td>
<td>0·6</td>
<td>—</td>
</tr>
<tr>
<td>Horizontal drilling</td>
<td>0·2</td>
<td>—</td>
<td>0·8</td>
<td>—</td>
<td>—</td>
<td>0·6</td>
<td>—</td>
</tr>
<tr>
<td>Boring</td>
<td>0·05</td>
<td>0·03</td>
<td>—</td>
<td>0·07</td>
<td>0·1</td>
<td>0·05</td>
<td>0·08</td>
</tr>
<tr>
<td>Planing</td>
<td>—</td>
<td>—</td>
<td>0·01</td>
<td>—</td>
<td>0·05</td>
<td>—</td>
<td>0·05</td>
</tr>
</tbody>
</table>

In the present month (January) and the five subsequent months certain machines will be down for maintenance. These machines will be:

January 1 grinder
February 2 horizontal drills
March 1 borer
April 1 vertical drill
May 1 grinder and 1 vertical drill
June 1 planer and 1 horizontal drill
### Table 12.5

<table>
<thead>
<tr>
<th>Retailer</th>
<th>Oil market (10^4 gallons)</th>
<th>Delivery points (10^4 gallons)</th>
<th>Spirit market (10^4 gallons)</th>
<th>Growth category</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>9</td>
<td>11</td>
<td>34</td>
<td>A</td>
</tr>
<tr>
<td>M2</td>
<td>13</td>
<td>47</td>
<td>411</td>
<td>A</td>
</tr>
<tr>
<td>M3</td>
<td>14</td>
<td>44</td>
<td>82</td>
<td>A</td>
</tr>
<tr>
<td>M4</td>
<td>17</td>
<td>25</td>
<td>157</td>
<td>B</td>
</tr>
<tr>
<td>M5</td>
<td>18</td>
<td>15</td>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>M6</td>
<td>19</td>
<td>26</td>
<td>183</td>
<td>A</td>
</tr>
<tr>
<td>M7</td>
<td>23</td>
<td>26</td>
<td>14</td>
<td>B</td>
</tr>
<tr>
<td>M8</td>
<td>21</td>
<td>54</td>
<td>215</td>
<td>B</td>
</tr>
<tr>
<td>M9</td>
<td>9</td>
<td>18</td>
<td>102</td>
<td>B</td>
</tr>
<tr>
<td>M10</td>
<td>11</td>
<td>51</td>
<td>21</td>
<td>A</td>
</tr>
<tr>
<td>M11</td>
<td>17</td>
<td>20</td>
<td>54</td>
<td>B</td>
</tr>
<tr>
<td>M12</td>
<td>18</td>
<td>105</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>M13</td>
<td>18</td>
<td>7</td>
<td>6</td>
<td>B</td>
</tr>
<tr>
<td>M14</td>
<td>17</td>
<td>16</td>
<td>96</td>
<td>B</td>
</tr>
<tr>
<td>M15</td>
<td>22</td>
<td>34</td>
<td>118</td>
<td>A</td>
</tr>
<tr>
<td>M16</td>
<td>24</td>
<td>100</td>
<td>112</td>
<td>B</td>
</tr>
<tr>
<td>M17</td>
<td>36</td>
<td>50</td>
<td>535</td>
<td>B</td>
</tr>
<tr>
<td>M18</td>
<td>43</td>
<td>21</td>
<td>8</td>
<td>B</td>
</tr>
<tr>
<td>M19</td>
<td>6</td>
<td>11</td>
<td>53</td>
<td>B</td>
</tr>
<tr>
<td>M20</td>
<td>15</td>
<td>19</td>
<td>28</td>
<td>A</td>
</tr>
<tr>
<td>M21</td>
<td>15</td>
<td>14</td>
<td>69</td>
<td>B</td>
</tr>
<tr>
<td>M22</td>
<td>25</td>
<td>10</td>
<td>65</td>
<td>B</td>
</tr>
<tr>
<td>M23</td>
<td>39</td>
<td>11</td>
<td>27</td>
<td>B</td>
</tr>
</tbody>
</table>

#### 12.14 Opencast Mining

A company has obtained permission to opencast mine within a square plot 200 ft × 200 ft. The angle of slip of the soil is such that it is not possible for the sides of the excavation to be steeper than 45°. The company has obtained estimates for the value of the ore in various places at various depths. Bearing in mind the restrictions imposed by the angle of slip the company decides to consider the problem as one of the extracting of rectangular blocks. Each block has horizontal dimensions 50 ft × 50 ft and a vertical dimension of 25 ft. If the blocks are chosen to lie above one another, as illustrated in vertical section in Figure 12.4, then it is only possible to excavate blocks forming an upturned pyramid. (In a three-dimensional representation Figure 12.4 would show four blocks lying above each lower block.)

If the estimates of ore value are applied to give values (in percentage of pure metal) for each block in the maximum pyramid which can be extracted then the following values are obtained:

<table>
<thead>
<tr>
<th></th>
<th>Level 1 (surface)</th>
<th>Level 2 (25 ft depth)</th>
<th>Level 3 (50 ft depth)</th>
<th>Level 4 (75 ft depth)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.5 1.5 1.5 0.75</td>
<td>1.5 2.0 1.5 0.75</td>
<td>0.75 0.75 0.5 0.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The cost of extraction increases with depth. At successive levels the cost of extracting a block is:

- Level 1: £3000
- Level 2: £6000
- Level 3: £8000
- Level 4: £10 000

The revenue obtained from a '100% value block' would be £200 000. For each block here the revenue is proportional to ore value.

Build a model to help decide the best blocks to extract. The objective is to maximize revenue – cost.

#### 12.15 Tariff Rates (Power Generation)

A number of power stations are committed to meeting the following electricity load demands over a day:

- 12 p.m. to 6 a.m.: 15 000 megawatts
- 6 a.m. to 9 a.m.: 30 000 megawatts
- 9 a.m. to 3 p.m.: 25 000 megawatts
- 3 p.m. to 6 p.m.: 40 000 megawatts
- 6 p.m. to 12 p.m.: 27 000 megawatts