Chapter 1

THE FORMULATION AND SOLUTION OF DISCRETE OPTIMISATION MODELS

H. Paul Williams
Department of Operational Research
London School of Economics, London
United Kingdom
h.p.williams@lse.ac.uk

Abstract This introductory chapter first discusses the applicability of Discrete Optimisation and how Integer Programming is the most satisfactory method of solving such problems. It then describes a number of modelling techniques, such as linearisng products of variables, special ordered sets of variables, logical conditions, disaggregating constraints and variables, column generation etc. The main solution methods are described, i.e. Branch-and-Bound and Cutting Planes. Finally alternative methods such as Lagrangian Relaxation and non-optimising methods such as Heuristics and Constraint Satisfaction are outlined.

Keywords: Integer Programming, Global Optima, Fixed Costs,Convex Hull, Reformulation, Presolve, Logic, Constraint Satisfaction.

1. The Applicability of Discrete Optimisation

The purpose of this introductory chapter is to give an overview of the scope for discrete optimisation models and how they are solved. Details of the modelling necessary is usually problem specific. Many applications are covered in other chapters of this book. A fuller coverage of this subject is given in [25] together with many references.

For solving discrete optimization models, when formulated as (linear) Integer Programmes (IPs), much fuller accounts, together with extensive references, can be found in Nemhauser and Wolsey [19] and Williams [24]. Our purpose, here, is to make this volume as self contained as possible by describing the main methods.

Limiting the coverage to *linear* IPs should not be seen as too restrictive in view of the reformulation possibilities described in this chapter.

It is not possible, totally, to separate modelling from the solution methods. Different types of model will be appropriate for different methods. With some methods it is desirable to modify the model in the course of optimisation. These two considerations are addressed in this chapter.

The modelling of many *physical* systems is dominated by *continuous* (as opposed to *discrete*) mathematics. Such models are often a simplification of reality, but the discrete nature of the real systems is often at a microscopic level and continuous modelling provides a satisfactory simplification. What's more, continuous mathematics is more developed and unified than discrete mathematics. The calculus is a powerful tool for the optimisation of many continuous problems. There are, however, many systems where such models are inappropriate. These arise with physical systems (e.g. construction problems, finite element analysis etc) but are much more common in decision making (operational research) and information systems (computer science). In many ways, we now live in a 'discrete world'. Digital systems are tending to replace analogue systems.

2. Integer Programming

The most common type of model used for discrete optimisation is an **Integer Programme** (IP) although Constraint Logic Programmes (discussed in the chapter by Hooker) are also applicable (but give more emphasis to obtaining feasible rather than optimal solutions). An IP model can be written:

$$\text{Maximise/Minimise} \quad \mathbf{c}'\,\mathbf{x} + \mathbf{d}'\,\mathbf{y} \tag{1.1}$$

$$\text{Subject to:} \quad A\,\mathbf{x} + B\,\mathbf{y} \leq \mathbf{b} \tag{1.2}$$

$$\mathbf{x} \in \Re, \mathbf{y} \in Z \tag{1.3}$$

x are (generally non-negative) continuous variables and **y** are (generally non-negative and bounded) integer variables.

If there are no integer variables we have a **Linear Programme** (LP). If there are no continuous variables we have a **Pure IP** (PIP). In contrast the above model is known as a **Mixed IP** (MIP). Most practical models take this form. It should be remarked that although the constraints and objective of the above model are linear this is not as restrictive, as it might seem. In IP, non-linear expressions can generally be linearised (with possibly some degree of approximation) by the addition of extra integer variables and constraints. This is explained in this chapter. Indeed IP models provide the most satisfactory way of solving, general non-linear optimisation problems, in order to obtain a **global** (as opposed to local) **optimum**. This again is explained later in this chapter.

LPs are 'easy' to solve in a well defined sense (they belong to the complexity class P). In contrast, IPs are difficult, unless they have a special structure, (they are in the NP – complete complexity class). There is often considerable scope for formulating, or reformulating, IP models in a manner which makes them easier to solve.

3. The Uses of Integer Variables

The most obvious use of integer variables is to model quantities which can only take integer values e.g. numbers of cars, persons, machines etc. While this may be useful, if the quantities will always be small integers (less than, say, 10), it is not particularly common. Generally such quantities would be represented by continuous variables and the solutions rounded to the nearest integer.

3.1 0-1 Variables

More commonly integer variables are restricted to two values, 0 or 1 and known as **0-1 variables** or **binary variables**. They then represent Yes/No decisions (e.g. investments). Examples of their use are given below. Before doing this it is worth pointing out that any bounded integer variable can be represented as a sum of 0-1 variables. The most compact way of doing this is to use a binary expansion of the coefficients. Suppose e.g. we have an integer variable y such that:

$$0 \leq y \leq U \tag{1.4}$$

Where U is an upper bound on the value of y. Create 0-1 variables:

$$y_0 \, , \, y_1 \, , \, y_2 \, , \quad \cdot \quad \cdot \quad \cdot \quad y_{\lfloor \log_2 U \rfloor}$$

then y can be represented by:

$$y_0 + 2y_1 + 4y_2 + \ldots.. + 2^{\lfloor \log_2 U \rfloor} y_{\lfloor \log_2 U \rfloor} \tag{1.5}$$

3.1.1 Fixed Charge Problems. A very common use of 0-1 variables is to represent investments or decisions which have an associated **fixed cost** (f). If carried out, other continuous variables $x_1, x_2, \ldots, x_n$ bringing in e.g. **variable profits** $p_1, p_2, \ldots, p_n$ come into play. This situation would be modelled as:

$$\text{Maximise} \qquad \sum_j p_j x_j - f\,y \tag{1.6}$$

$$\text{Subject to:} \quad \sum_j x_j - M\,y \quad \leq 0 \tag{1.7}$$

(and other constraints)

where M represents an upper bound on the combined level of the continuous activities.

If $y = 0$ (investment not made) all the $x_j$are forced to zero. On the other hand if $y = 1$ (investment is made) the fixed cost f is incurred but the x_j can take positive values and profit is made.

There are, of course, many extensions to this type of model where alternative or additional investments can be made, etc. The basic idea is, however, the same.

3.1.2 Indicator Variables. A very powerful modelling technique is to use 0-1 variables to distinguish situations which are ('discretely') different. If there is a simple dichotomy then the two possible values of a 0-1 variable can be used to make the distinction. For example, suppose we want to model the condition

$$\sum_j a_{1j}x_j \le b_1 \quad \textbf{or} \quad \sum_j a_{2j}x_j \le b_2 \tag{1.8}$$

Such a condition is known as a *disjunction* of constraints (in contrast to the usual LP *conjunction* of constraints). Let $y = 0$ impose the first constraint in the disjunction and y= 1 impose the second constraint. This is achieved by the following *conjunction of* constraints

$$\sum_j a_{1j}x_j - M_1 y \le b_1 \tag{1.9}$$

$$\sum_j a_{2j}x_j + M_2 y \le M_2 + b_2 \tag{1.10}$$

M_1 is an upper bound on the value of the expression $\sum\limits_j a_{1j}x_j - b_1$ and M_2 an upper bound on $\sum\limits_j a_{2j}x_j - b_2$. (If either of the expressions has no upper bound then the disjunction can only be modelled in special conditions.) There is an alternative and preferable way of modelling this condition which is considered later in this chapter.

Should a disjunction of more than two constraints be needed then more than one $0-1$ variable can be used. For example the situation

$$\sum_j a_{ij}x_j \le b_1 \ \textbf{or} \ \sum_j a_{2j}x_j \le b_2 \ \textbf{or} \ldots \textbf{or} \ \sum_j a_{nj}x_j \le b_n \tag{1.11}$$

can be modelled as

$$\sum_j a_{ij}x_j - M_1 y_1 \le b_1 \tag{1.12}$$

$$\sum_j a_{2j}x_j - M_2 y_2 \le b_2 \tag{1.13}$$

$$\sum_j a_{nj}x_j - M_n y_n \leq b_n \tag{1.14}$$

$$y_1 + y_2 + \cdots + y_n \leq n - 1 \tag{1.15}$$

The situation where at least k of a given set of constraints $(1 < k < n)$ hold can easily be modelled by amending (1.15).

There are many situations where one wishes to model such disjunctions. For example:

- This operation must finish before the other starts **or** vice versa;
- We must either not include this ingredient **or** include it above a given threshold level.
- At least (most) k of a given set of possible warehouses should be built.

In fact all of IP could be reduced to disjunctions of linear constraints. This is only a useful paradigm for some problems. However the modelling of disjunctive conditions has been thoroughly analysed in Balas [2].

3.2 Special Ordered Sets of Variables (Discrete Entities)

If a quantity can take a number of discrete values then a convenient method of modelling is a *Special Ordered Set of Type 1* (S1 set). Suppose we wish to model a quantity which can take values $a_1, a_2, \ldots, a_n$. We would represent this quantity by

$$x = a_1y_1 + a_2y_2 + \cdots + a_ny_n \tag{1.16}$$

and with the constraint

$$y_1 + y_2 + \cdots + y_n = 1$$

with the stipulation that

Exactly one member of the set $\{y_1, y_2, \cdots, y_n\}$ *can be non-zero*

Note that it is not necessary to stipulate that the y_j variables be integer and 0-1.

An S1 set (as opposed to the individual variables in it) should be regarded as a *discrete entity*. It can be considered as a generalisation of a general integer variable where values are not evenly spaced. S1 sets are regarded as entities analogous to integer variables when the Branch-and Bound algorithm, described later in this chapter, is used.

A variant of S1 sets are *Special Ordered sets of Type 2* (S2 sets). These are widely used to model *non-linear functions*. In order to model a non-linear function it must be separated into the sum of non-linear functions of a single variable. This can generally be done by introducing new variables and constraints. (The possibility of doing this goes back to one of Hilbert's problems.)

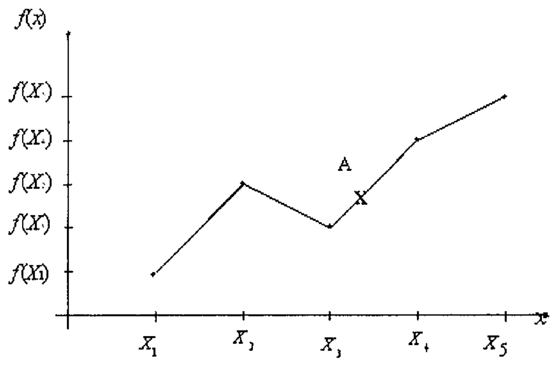

Figure 1.1. A piecewise linear approximation to a non-linear function

Each non-linear function of a single variable is approximated by a *piecewise linear function* i.e. a series of linear segments as illustrated in Figure 1.1. A grid value is defined for the relevant values of the argument x (not necessarily evenly spaced) and the function $f(x)$ defined at those values. We define an S2 set $\{y_1, y_2, \dots y_5\}$ with the constraints

$$x = X_1 y_1 + X_2 y_2 + X_3 y_3 + X_4 y_4 + X_5 y_5 \tag{1.17}$$

$$f(x) = f(X_1)y_1 + f(X_2)y_2 + f(X_3)y_3 + f(X_4)y_4 + f(X_5)y_5 \tag{1.18}$$

$$y_1 + y_2 + y_3 + y_4 + y_5 = 1 \tag{1.19}$$

together with the stipulation that

At most two adjacent y_5 can be non-zero

Note that the y_j are continuous. If, for example, $y_3 = \frac{1}{4}, y_4 = \frac{3}{4}$ (implying the other y_j=0) this corresponds to the point A in Figure 1.1. By this means, only the points on one of the straight-line segments in Figure 1.1 are represented by$(x, f(x))$. Clearly a more refined grid can give a more accurate approximation to a function (at the expense of more variables).

The concept of Special Ordered Sets are described by [3] with an extension to deal with functions of more than one variable in [4]. There are, of course, alternative methods of modelling condition (1.11) using 0-1 variables since we again have a disjunction of possibilities. However, using Special Ordered Sets has computational advantages when using the Branch-and-Bound algorithm.

4. The Modelling of Common Conditions

4.1 Products of Variables

It has already been remarked that non-linear expressions can be reformulated using linear constraints involving integer variables. A common non-linearity is products of integer variables. As explained above (bounded) general integer variables can always be expressed using 0-1 variables. Therefore we confine our attention to products of 0-1 variables.

Consider the product $y_1 y_2$ which we will represent by a new variable z where y_1 and y_2 are 0-1 variables. This product itself can only take values 0 or 1 depending on the values of y_1 and y_2. If either is zero the product itself is 0. Otherwise it is 1.

Logically we have

$$y_1 = 0 \ \textbf{or} \ \ y_2 = 0 \ \text{(or both)} \ \textbf{implies} \ \ z = 0$$

This can be modelled by

$$y_1 + y_2 - 2z \geq 0 \tag{1.20}$$

(There is a 'better' way of modelling this which will be given later in this chapter.) We also have to model

$$y_1 = 1 \ \ \textbf{and} \ \ \ y_2 = 1 \ \ \textbf{implies} \ \ z = 1.$$

This can be modelled by

$$y_1 + y_2 - z \leq 1 \tag{1.21}$$

If we have a product of 3 or more $0-1$ variables then we can repeat the above formulation procedure.

A product of a continuous variable x and a 0-1 variable y can be represented by a continuous variable z. The variable z will then be equal to x or to zero depending on whether y is 1 or 0. Either way we have

$$z - x \leq 0 \tag{1.22}$$

If $y = 1$ then we want the condition $z = x$. This may be done by modelling the condition

$$y = 1 \ \ \textbf{implies} \ \ z \geq x$$

since, together with (1.22), this would imply $z = x$. We can model the condition as

$$x - z + My \leq M \tag{1.23}$$

M must be chosen as a suitably large number (an upper bound) which x (and therefore z) will not exceed.

If we wish to model the product of an 0-1 variable and an *expression* this formulation can be extended in an economical manner, as done by Glover [10].

4.2 Logical Conditions

The common use of 0-1 variables in IP models suggests the analogy with modelling of True/False propositions which forms the subject matter of the Propositional Calculus (Boolean Algebra). This close relationship is explored further in this book in Chapter 3 by Hooker. Here we give a taste for the power of IP to model the relationships of the Propositional Calculus.

In IP (and LP) our propositions are constraints which may be true or false. (An LP model consists of a conjunction of constraints i.e. they must all be true). We can represent the truth or falsity of a constraint by the setting of a 0-1 variable. If, for example, we have the constraint

$$\sum_j a_j\ x_j \le b \tag{1.24}$$

then we can incorporate the 0-1 variable y into the constraint to give

$$\sum_j a_j x_j + My \le M + b \tag{1.25}$$

where M is an upper bound on the expression

$$\sum_j a_j x_j - b \tag{1.26}$$

If $y = 1$ the constraint (1.24) is forced to hold. If $y = 0$ it becomes vacuous. Sometimes we may wish to model the condition

$$\sum_j a_j x_j \le b \ \textbf{implies} \quad y = 1$$

This can be represented by its equivalent *contrapositive* statement

$$y = 0 \ \textbf{implies} \ \sum_j a_j x_j > b \tag{1.27}$$

Since it is conventional only to use non-strict inequalities we can replace

$$\sum_j a_j x_j > b$$

by

$$\sum_j a_j x_j \ge b + \epsilon \tag{1.28}$$

where ϵ is a suitable small positive number. Now (1.27) can be modelled (by analogy with (1.25)) as

$$\sum_j a_j x_j + my \ge b + \epsilon \tag{1.29}$$

where m is a lower bound on the expression

$$\sum_j a_j x_j - b$$

It is now possible to model the standard connectives of the Propositional Calculus applied to constraints by means of (in)equalities between 0-1 variables.

We have already modelled the '**or**' condition by constraints (1.9) and (1.10) and (1.12) to (1.15). The '**and**' condition is simply modelled by repeating the constraints (as in LP). The '**implies**' condition is modelled thus:

$$y_1 = 1 \ \textbf{implies} \ y_2 = 1$$

$$\text{is represented by } y_1 - y_2 \leq 0$$

We now have all the machinery for modelling logical conditions which will be explained in greater depth in Hooker's chapter (see chapter 3).

5. Reformulation Techniques

Despite advances in methods of solving Discrete Optimisation problems, as well as the dramatic increase in the speed and storage capacity of computers, many such problems still remain very difficult to solve. There is, however, often great flexibility in the way such problems are modelled as Integer Programmes. It is frequently possible to remodel a problem in a way which makes it easier to solve. This is the subject of this section.

5.1 The Convex Hull of Integer Solutions

The constraints of a Linear Programme (LP) restrict the feasible solutions to a set which can be represented by a Polytope in multidimensional space. For Integer Programmes the set is further restricted to the lattice of integer points within a Polytope. This is illustrated, in Figure 1.2, by the constraints of a 2-Variable IP, which can therefore be represented in 2-dimensional space.

$$-x_1 + 2x_2 \leq 7 \quad (1.30)$$

$$x_1 + 3x_2 \leq 15 \quad (1.31)$$

$$7x_1 - 3x_2 \leq 23 \quad (1.32)$$

$$x_1, x_2 \geq 0 \text{ and integer} \quad (1.33)$$

The boundaries of constraints (1.30), (1.31) and (1.32) are represented by the lines AB, BC and CD respectively and the non-negativity conditions by OA and OD. However, the integrality restrictions on x_1 and x_2 restrict us further to the lattice of points marked. If we were to ignore the integrality conditions then, given an objective function, we would have an LP model. An optimal

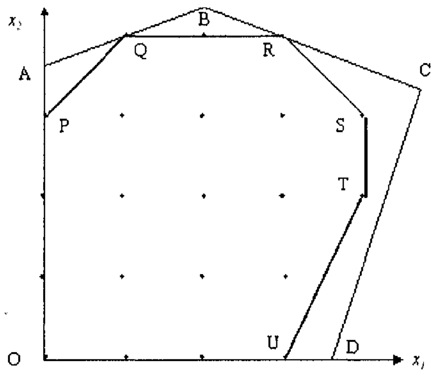

Figure 1.2. The convex hull of a pure IP

solution would lie at one of the vertices O, A, B, C, or D (if there were alternative solutions some of them would be between these vertices). The LP got by ignoring the integrality conditions is known as the "LP Relaxation" of the model.

However, if we were to represent the constraints whose boundaries are the bold lines we would have a more constrained LP problem whose solutions would be O, P, Q, R, S, T or U which would satisfy the integrality requirements. The region enclosed in the bold lines is known as the *Convex Hull of Feasible Integer Solutions.* It is the smallest convex set containing the feasible integer solutions. For this example the constraints defining the convex hull are:

$$-x_1 + x_2 \leq 3 \tag{1.34}$$

$$x_2 \leq 4 \tag{1.35}$$

$$x_1 + x_2 \leq 7 \tag{1.36}$$

$$x_1 \leq 4 \tag{1.37}$$

$$2x_1 - x_2 \leq 6 \tag{1.38}$$

$$x_1, x_2 \geq 0 \tag{1.39}$$

Computationally LP models are much easier to solve than IP models. Therefore reformulating an IP by appending or substituting these *facet defining constraints* would appear to make a model easier to solve. With some types of model it is fairly straightforward to do this. In general, however, the derivation of facet defining constraints is extremely difficult and there is no known systematic way of doing it. What is more, there are often an astronomic number of facets making it impossible to represent them all with limited computer storage.

Before describing ways of partially using the concept above, to our advantage, we point out that the example above is a Pure Integer Programme. The concept still applies to Mixed Integer Programmes. Suppose, for example, that constraints (1.33) have been modified to only stipulate that x_2 should be integer. The feasible solutions would be as represented by the horizontal lines in Figure 1.3. The convex hull of feasible integer solutions is described by the

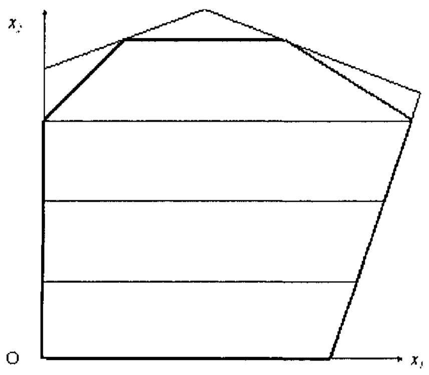

Figure 1.3. The convex hull of a mixed IP

bold lines and the facet defining constraints are:

$$\begin{aligned} -x_1 + x_2 &\leq 3 \quad (1.40)\\ x_2 &\leq 4 \quad (1.41)\\ 7x_1 + 11x_2 &\leq 65 \quad (1.42)\\ 7x_1 - 3x_2 &\leq 23 \quad (1.43)\\ x_1, x_2 &\geq 0 \quad (1.44) \end{aligned}$$

We now describe ways in which some commonly arising constraints can be reformulated to make the associated LP Relaxation 'tighter', even if we do not produce facet defining constraints for the complete model.

5.2 'Big M' coefficients

These arise when a particular set of activities (continuous or integer) can only take place if another, $0-1$ variable, takes the value 1. An example of this is the constraint (1.7) in the Fixed Charge problem. The condition is correctly modelled so long as the value of M is sufficiently large not to place a spurious restriction on the value of $\sum_j x_j$ (or some other set of activities). But if M is larger than it need be, the associated LP relaxation will be less constrained than it need be i.e. the associated polytype of the LP relaxation will be larger than necessary. Therefore when such a situation arises it is desirable to make M a strict upper bound on the appropriate expression, if possible. It may be worth maximising the appropriate expression subject to the other constraints (as an LP) in order to find as small a value of M to be acceptable. A number of packages try to reduce the values of 'Big M coefficients' automatically.

5.3 Disaggregating Constraints

A common PIP constraint takes the form

$$x_1 + x_2 + \ldots + x_n - ny \leq 0 \tag{1.45}$$

where all variables are $0 - 1$. This models the condition that if any of x_j take the value 1 then so must y. An analogous constraint is

$$x_1 + x_2 + \ldots + x_n - ny \geq 0 \tag{1.46}$$

which models the condition that if $y = 1$ then all of the x_j must be one. Constraint (1.20) is a special case of this. Both (1.45) and (1.46) can be disaggregated into

$$x_j - y \leq 0 \quad \text{for all} \quad j \tag{1.47}$$

and

$$x_j - y \geq 0 \quad \text{for all} \quad j \tag{1.48}$$

respectively. The groups of constraints (1.47) and (1.48) both have the same set of feasible integer solutions as (1.45) and (1.46) respectively but have more constrained LP relaxations. e.g. the solution $x_1 = \frac{1}{2}, x_2 = \frac{3}{4}, x_3 = \ldots = x_n = 0, y = \frac{1}{n}$ satisfies (1.45) and (1.46) but violates (1.47) and (1.48). Hence this fractional solution cannot arise from an LP relaxation using (1.47) and (1.48). However, any fractional solution which satisfies (1.47) and (1.48) also satisfies (1.45) and (1.46) respectively. There are many other, more complicated, constraints and groups of constraints which have been analysed to give tighter LP relaxations. See for example Wolsey [26] and Nemhauser and Wolsey [19].

5.4 Splitting Variables

In order to disaggregate constraints it is sometime possible to split variables up into component variables which can then appear in different constraints. A general way of doing this is described by Balas [2] and Jeroslow [15] under the title 'Disjunctive Constraints'.

We have already shown how it is possible to model a disjunction of constraints (1.11) by (1.12) to (1.15). There is another formulation for which the LP relaxation is tighter.

Suppose we have a disjunction of groups of constraints which can be written as

$$A_1\boldsymbol{x}\geq\boldsymbol{b}_1 \textbf{ or } A_2\boldsymbol{x}\geq\boldsymbol{b}_2 \textbf{ or} \ldots \textbf{ or } A_n\boldsymbol{x}\geq\boldsymbol{b}_n \tag{1.49}$$

where $\boldsymbol{x}$ and $\boldsymbol{b}_i$ are vectors. We also assume that the constraints in each of the 'clauses' above have the same *recession directions.* This means that if they are open polytopes, as represented in Figure 4, the directions of the extreme rays are the same. The polytopes in Figure 4 have different recession directions since direction DS does not form part of the left hand polytope. If, as is the

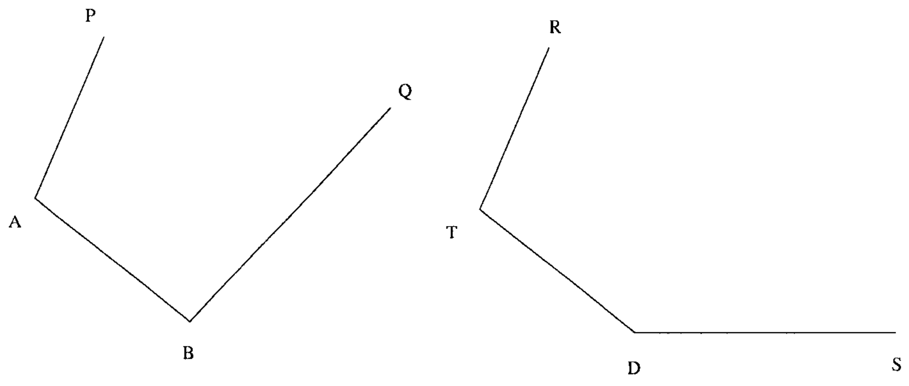

Figure 1.4. Polytopes with different recession directions

case with the constraints in (1.11), the polytopes are all closed then the problem does not arise since there are no recession directions. In practice this will often be the case as we will place 'big M' coefficients as upper bounds on the variables. For disjunction (1.49), however, we make no such assumption. In order to model (1.49) we split the vector $\boldsymbol{x}$ into n component vectors $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_n$ i.e.

$$\boldsymbol{x} = \boldsymbol{x}_1+\boldsymbol{x}_2+\ldots+\boldsymbol{x}_n \tag{1.50}$$

We then specify the following constraints

$$A_j\boldsymbol{x}_j \quad \geq \boldsymbol{b}_j y_j \text{ for all } j \tag{1.51}$$

$$y_1 + y_2 + \ \ldots \ + y_n = 1 \tag{1.52}$$

where y_j are $0-1$ integer variables. Balas [2] shows that if it is possible to represent an IP as a disjunction of polytopes in this way then we can obtain a convex hull representation (in the higher dimensional space of $\boldsymbol{x} = \boldsymbol{x}_1+\boldsymbol{x}_2+\ldots+\boldsymbol{x}_n$)

5.5 Multicommodity Flows

If a MIP model contains variables representing flows of material it may be advantageous to disaggregate these flows into different commodities. This is illustrated by the **lot sizing problem** (see e.g. Wolsey [26]). In this problem it has to be decided, in different periods, whether to produce a certain quantity (a 'lot') of material to satisfy future demand. There is a fixed charge associated with producing any at all. Hence an optimal policy will be only to produce in certain periods but retain stock from this production to satisfy some of the demand in the future.

Let x_i = Quantity produced in period i
$y_i = 1$ if $x_i \geq 0$
$= 0$ otherwise
z_i = Stock held at the end of the period i
d_i = Demand in period i
f_i = Fixed cost of production in period i

One possible way of modelling the constraints is

$$x_i - M_i y_i \leq 0 \tag{1.53}$$

$$x_1 - z_1 = d_1 \tag{1.54}$$

$$z_{i-1} + x_i - z_i = d_i \quad \text{for all} \quad i \neq 1 \tag{1.55}$$

Notice the use of the 'big-M' coefficient in constraints (1.53) representing upper bounds on the amount produced in each period.

An alternative, and more constrained, way of modelling this problem is to split the variables into components representing different 'commodities'. We distinguish between the different quantities produced in a period to meet demand in *different* future periods.

Let x_{ij} = Quantity produced in period i to meet demand in period j

z_{ij} = Stock held at end of period i to meet demand in period j

The constraints can now be reformulated as

$$\begin{aligned} x_{ij} - d_j y_i &\leq 0 \text{ for all } i, j & (1.56)\\ x_{1j} - z_{1j} &= d_j \text{ for all } j & (1.57)\\ \sum_{i<j} (z_{i-1,j} + x_{ij} - z_{ij}) &= d_j \text{ for all } j & (1.58) \end{aligned}$$

Notice that it is no longer necessary to use the 'big-M' coefficients since the demand in period j places an upper bound on the value of each x_{ij} variable.

If all the constraints were of the form (1.56), (1.57) and (1.58) it can be shown that the LP relaxation would produce the integer optimal solution. In practice there will probably be other constraints in the model which prevent this happening. However, this second formulation is more constrained (although larger).

5.6 Reducing the Number of Constraints

One of the effects of most of the reformulations described in the previous section has been to increase the number of constraints. While there is great advantage to be gained in tightening the LP relaxation in this way the increased size of the models will, to some extent, counter the reduction in overall solution time. Therefore, in this section, we look at ways of reducing the number of constraints in an IP model.

5.6.1 Avoiding Constraints by Virtue of Optimality. Sometimes certain constraints will automatically be satisfied at the optimal solution. For example, with the formulation of the fixed charge problem constraint (1.7) does not rule out the 'stupid' solution $x_j = 0$ for all j and $y = 1$. However, it is not necessary to put in extra constraints to avoid this since such a solution would be non-optimal.

However, there are situations in which it is desirable to put in constraints which are not necessary, by virtue of optimality, but which aid the solution process using the Branch and Bound algorithm described below. Hooker et al [14] discusses such an example in relation to modelling a chemical process and terms the extra constraints as 'logic cuts'.

5.6.2 Avoiding Constraints by Introducing Extra Variables. A number of formulations of important Combinatorial Optimisation problems involve an **exponential** number of constraints. The best known example of this is the famous **Travelling Salesman Problem** discussed in eg Lawler et al [17]. The problem is to route a Salesman around a number of cities, returning to the beginning and covering the minimum total distance. It can be formulated as a PIP model as follows:

$$\begin{aligned} x_{ij} &= 1 \quad \text{if tour goes from } i \text{ to } j \text{ directly} \\ &= 0 \quad \text{otherwise} \end{aligned}$$

Given that the distance from i to j is c_{ij} the model is

$$\text{Minimise} \quad \sum_{ij} c_{ij} x_{ij} \tag{1.59}$$

$$\text{subject to} \quad \sum_i x_{ij} = 1 \text{ for all } j \tag{1.60}$$

$$\sum_j x_{ij} = 1 \text{ for all } i \tag{1.61}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \text{for all } S \subset \{2, 3, \ldots, n\} \tag{1.62}$$

where $2 \leq |S| \leq |V| - 2$. Constraints (1.60) and (1.61) guarantee that each city is entered exactly once and left exactly once respectively. However, constraints (1.62) are needed to prevent 'subtours' i.e. going around disconnected subsets of the cities. There are $2^{n-1} - n - 1$, such constraints: i.e. an **exponential** number. It would be impossible to fit all these constraints in a computer for even modest values of n. In practice such constraints are usually added (with others) on an 'as-needed' basis in the course of optimisation.

However, it is possible to avoid an exponential number of constraints by introducing extra variables (a polynomial number). There are a number of ways of doing this which are surveyed by Orman and Williams [20]. We present one such formulation here. It relies on a Network Flow formulation associated with the network of arcs between cities.

$$y_{ij} = \text{Flow (of some commodity) between } i \text{ and } j$$

$$y_{ij} - (n-1)\, x_{ij} \leq 0 \quad \text{for all } i, j \tag{1.63}$$

$$\sum_j y_{1j} = n - 1 \tag{1.64}$$

$$\sum_j y_{ij} - \sum_k y_{ki} = -1 \quad \text{for all } i \neq 1 \tag{1.65}$$

together with constraints (1.60) and (1.61).

Constraints (1.63) only allow flow in an arc if it exists (similar to the 'big M' constraints discussed earlier). Constraint (1.64) introduces a flow of $n - 1$ units of the commodity into city 1 and constraint (1.65) takes 1 unit of the commodity out of all the other $(n - 1)$ arcs. In order to dispose of all the commodity the underlying network must be connected, so ruling out subtours.

Hence the exponential number of constraints (1.62) has been replaced by constraints (1.63), (1.64) and (1.65): a total of n^2. In addition we have doubled the number of variables from $n\,(n-1)$ to $2n\,(n-1)$.

Unfortunately this formulation does not perform as well as the conventional formulation, for reasons described below, if a linear programming based IP algorithm is used. Rather surprisingly this formulation can be drastically improved by a *multicommodity* flow formulation analogous to that described for the lot sizing problem. This results in a model whose LP relaxation has equal strength to the conventional formulation, but has only a polynomial number of constraints.

5.7 Column Generation

Besides the advantages of introducing extra variables to avoid an excess number of constraints it may be very difficult (or impossible) to capture some complicated conditions by means of (linear) constraints. In such circumstances it may only be practical to enumerate all, or some of, the possible solutions to the complicating conditions. These can then be generated in the course of optimization if deemed worthwhile.

The classic example of this is the Cutting Stock Problem (see e.g. Gilmore and Gomory [9]) where it is wished to cut given orders for widths of material (e.g. wallpaper) out of standard widths. The objective is to minimise total waste (which is equivalent to minimising the total number of rolls used). An equivalent problem is sometimes referred to as the 'Bin Packing Problem'. Here we wish to pack items into bins using the minimum number of bins.

A possible formulation would be to include variables with the following interpretation:

$$x_{ij} = \text{Quantity of order } i \text{ (e.g. a given width) taken from roll } j$$

Constraints could then be stated to:

(i) limit the widths taken from each standard roll to be within the standard width

(ii) meet total demand

This would be a somewhat cumbersome model with a large amount of unnecessary symmetry (i.e. many equivalent solutions would be investigated in the course of optimisation).

The usual way of solving such problems is to enumerate some of the solutions to (i). This is done by giving possible patterns of widths which may fit into the standard widths.

We illustrate this important technique by a small example. A plumber has a stock of 13 metre copper pipes and needs to cut off the following orders:

10 lengths of 4 metres
10 lengths of 5 metres
23 lengths of 6 metres

How should he cut up the 13 metre pipes in order to minimise the number he needs to use ? There are a number of possible cutting patterns he could adopt,

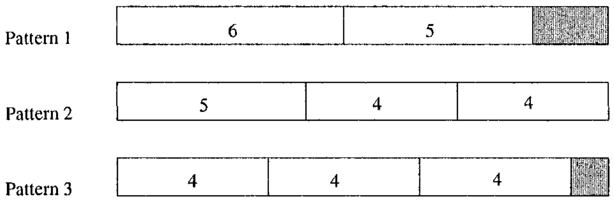

Figure 1.5. A cutting pattern

for example three are shown in Figure 1.5. In fact, in this example, there are only six possible patterns (leaving out obviously redundant ones, which are 'dominated' by others). However we will begin by restricting ourselves to the above three.

Variables are introduced with the following interpretation:

$x_j =$ Number of standard pipes cut using pattern j in our model

The model becomes:

$$\text{Minimise } x_1 + x_2 + x_3 \tag{1.66}$$

$$\begin{aligned} \text{Subject to:} \quad 2x_2 + 3x_3 &\geq 10 && (1.67)\\ x_1 + x_2 &\geq 10 && (1.68)\\ x_1 &\geq 23 && (1.69)\\ x_1, x_2, x_3 &\geq 0 \text{ and integer} \end{aligned}$$

Notice that we have modelled the demands for 4, 5 and 6 metre pipes respectively by constraints (1.67) to (1.69) i.e. we have modelled constraints (ii). However we have predefined some solutions to constraints (i). If we solve the above model we obtain the solution

$$x_1 = 23, x_2 = 0, x_3 = 4$$

i.e. we use 23 of pattern 1 and 4 of pattern 3 making a total of 27 standard lengths. However, we have not incorporated some patterns in the model. Notice that each possible pattern gives a *column* of the model e.g. the column for x_2 has coefficients 2,1,0 since pattern 2 has 2 of length 4, 1 of length 5 and 0 of length 5.

If the above model is solved as a *Linear Programme* (LP) we obtain *dual values* for the constraints (this concept is explained in any standard text on LP). In this case they are $\frac{1}{3}, 0, 1$. Suppose we were to consider another pattern with a_1 of length 4, a_2 of length 5 and a_3 of length 6 we must have

$$4a_1 + 5a_2 + 6a_3 \leq 13 \tag{1.70}$$

The variable corresponding to this pattern would have a *reduced cost* of

$$1 - \frac{1}{3}a_1 - a_3$$

To make it a desirable variable to enter the solution we wish to make this quantity as small as possible.

$$\text{i.e. Maximise} \quad \frac{1}{3}a_1 + a_3 \tag{1.71}$$

subject to constraint (1.70). This type of IP with one constraint is known as a *Knapsack Problem* and efficiently solved by *Dynamic Programming* (see e.g. Bellman [5]). Details of this procedure are beyond the scope of this chapter. For this example the optimal solution is $a_1, a_2 = 0, a_3 = 2$ i.e. we should consider using the pattern in Figure 1.6.

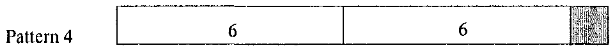

Figure 1.6. Optimal pattern

Note that the analysis has been based on the *LP Relaxation* and not the IP solution to (1.66)-(1.69). Therefore it is not necessarily the case that the new pattern will be beneficial. However, in this case we append a new variable x_4 to the model, representing pattern 4. This gives

$$\text{Minimise} \quad x_1 + x_2 + x_3 + x_4 \tag{1.72}$$

$$\begin{aligned} \text{Subject to:} \quad 2x_2 + 3x_3 &\geq 10 && (1.73)\\ x_1 + x_2 &\geq 10 && (1.74)\\ x_1 + 2x_4 &\geq 23 && (1.75)\\ x_1, x_2, x_3, x_4 &\geq 0 \text{ and integer} \end{aligned}$$

This revised model has optimal solution

$$x_1 = 5, \; x_2 = 5, \; x_3 = 0, \; x_4 = 9$$

i.e. we use 5 of pattern 1, 5 of pattern 2 and 9 of pattern 4 making a total of 19 standard lengths. Clearly this is an improvement of the previous solution. The procedure can be repeated generating new patterns only when worthwhile. In fact, for this example, this is the optimal solution demonstrating that only 3 of the six possible patterns are ever needed.

Column Generation techniques, such as this, are common in LP and IP to avoid excessive numbers of variables. The exact formulations of the applications are problem specific and beyond the scope of this chapter. One of the most economically important applications of the technique is to the Air Crew Scheduling problem.

6. Solution Methods

There is an important class of problems for which Linear Programming (LP) automatically gives integer solutions making specialist IP methods unnecessary. These are models which are presented as, or can be converted to, *Network Flow* models.

It is important to recognise that IP models are generally much more difficult to solve than LP models. Theoretically they lie in the NP-complete complexity class (unless they have a special structure) whereas LP models lie in the P complexity class (see for example Garey and Johnson [8]). This theoretical distinction is borne out by practical experience. Some IP models can prove very difficult to solve optimally in a realistic period of time. It is often much more difficult (than for LP models) to predict how long it will take to solve them. However, as discussed earlier, reformulation can sometimes have a dramatic effect on solution success.

Historically the first method used to solve IPs was known as '*Cutting Planes*' and is due to Gomory [11] and [13]. In practice, however, another, simpler, method known as Branch-and-Bound due to Land and Doig [16] and Dakin [7] was used and variants formed the basis of commercial systems. These two methods are described in the next two sections.

Commercial systems now, however, use a combination of both methods. While Branch-and-Bound forms the basic solution framework, Cutting Planes are used within this framework (Branch-and-Cut). This combined approach is also described here. In addition heuristic choices are made within Branch-and-Bound to guide the, somewhat arbitrary, tree search.

Reformulation methods are also incorporated in some systems (usually known as Presolve).

Finally variables (columns) are sometimes generated in the course of optimization. This is usually referred to as 'Branch-and-Price' for the reason described in the previous section.

6.1 Cutting Planes

In order to illustrate this approach we will refer to the example illustrated in figure 1.2. Each of the new facet defining constraints, given there (inequalities (1.34), (1.35), (1.36), (1.37) and (1.38)) can be regarded as *cutting planes*. If appended to the model they may aid the solution process. Rather than define them all, prior to solving the model, the usual approach is to append them, in the course of optimisation, if needed.

We illustrate this process by considering the model with an objective function i.e.

$$\text{Maximise } x_1 + x_2 \tag{1.76}$$

$$\text{Subject to: } -x_1 + 2x_2 \leq 7 \tag{1.77}$$

$$x_1 + 3x_2 \leq 15 \tag{1.78}$$

$$7x_1 - 3x_2 \leq 23 \tag{1.79}$$

$$x_1, x_2 \geq 0 \text{ and integer} \tag{1.80}$$

Figure 1.7 illustrates this. This first step is to solve the *Linear Programming*

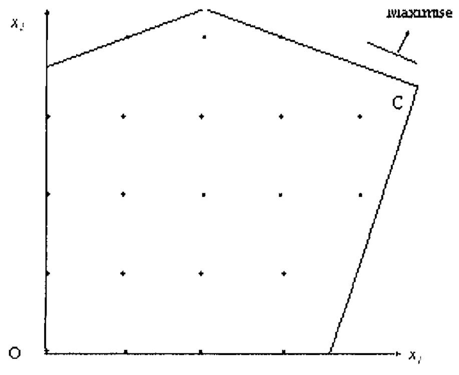

Figure 1.7. An integer programme

Relaxation of the model. That is to ignore the integrality requirements and to solve the model as an (easier) LP.

The concept of a '*Relaxation*' is broader than this and used widely in the solution of IP models. The essential idea

$$\text{Subject to:} \quad 2x_2 + 3x_3 \geq 10 \tag{1.81}$$

$$x_1 + x_2 \geq 10 \tag{1.82}$$

$$x_1 + 2x_4 \geq 23 \tag{1.83}$$

$$x_1, x_2, x_3, x_4 \geq 0 \text{ and integer}$$

is to leave out some of the conditions or constraints on a (difficult) IP model and solve the resultant (easier) model. By leaving out constraints we may well obtain a '*better*' solution (better objective value), but one which does not satisfy all the original constraints. This solution is then used to advantage in the subsequent process.

For the above example the optimal solution of the LP Relaxation is

$$x_1 = 4\frac{3}{4}, x_2 = 3\frac{5}{12}, \text{ Objective } = 8\frac{1}{6}$$

We have obtained the optimal fractional solution at C. The next step is to define a *cutting plane* which will *cut-off* the fractional solution at C, without removing any of the feasible integer solutions (represented by bold dots). This is known as the *separation* problem. Obviously a number of cuts are possible. The facet defining constraints, are possible cuts as are "shallower" cuts which are not facets. A major issue is to create a *systematic* way of generating those cuts (ideally facet defining) which *separate* C from the feasible integer solutions. Before we discuss this, however, we present a major result due to Chvátal [6]. This is that all the facet defining constraints (for a PIP) can be obtained by a finite number of repeated applications of the following two procedures.

(i) *Add together constraints* in suitable multiples (when all in the same form eg "$\leq$" or "$\geq$") and add or subtract "=" constraints in suitable multiples.

(ii) Divide through the coefficients by their greatest common divisor and *round the right-hand-side coefficient* up (in the case of "$\geq$" constraints) or down (in the case of "$\leq$" constraints).

We illustrate this by deriving all the facet defining constraints for the example above. However, we emphasise that our choice of which constraints to add, and in what multiples, is ad-hoc. It is a valid procedure but not systematic. This aspect is discussed later.

1. Add $-x_1 + 2x_2 \leq 7$

$$-x_1 \leq 0$$

to give $-2x_1 + 2x_2 \leq 7$

Divide by 2 and round to give

$$-x_1 + x_2 \leq \left\lfloor \frac{7}{2} \right\rfloor = 3 \tag{1.84}$$

2. Add $-x_1 + 2x_2 \leq 7$

$$-x_1 + 3x_2 \leq 15$$

to give $5x_2 \leq 22$

Divide by 5 and round to give

$$x_2 \leq \left\lfloor \frac{22}{5} \right\rfloor = 4 \tag{1.85}$$

3. Add $x_1 + 3x_2 \leq 15$

$$-7x - 3x_2 \leq 23$$

$$\text{to give } 8x_1 \leq 38$$

Divide by 2 and round to give

$$x_1 \leq \left\lfloor \frac{38}{8} \right\rfloor = 4 \tag{1.86}$$

4. Add $\quad x_1 + 3x_2 \leq 15$

$$x_1 \leq 4$$

$$\text{to give } 3x_1 + 3x_2 \leq 23$$

Divide by 3 and round to give

$$-x_1 + x_2 \leq \left\lfloor \frac{23}{3} \right\rfloor = 7 \tag{1.87}$$

5. Add $\quad 2 \text{ x } (7x_1 - 3x_2 \leq 23)$

$$-x_1 \leq 0$$

$$\text{to give } 14x_1 - 7x_2 \leq 46$$

Divide by 7 and round to give

$$-2x_1 - x_2 \leq \left\lfloor \frac{46}{7} \right\rfloor = 6 \tag{1.88}$$

Constraints (1.84) to (1.88), together with the original constraints (1.80) make up all the facet defining constraints for this model. Note that all the new facet defining constraints, apart from constraint (1.87), have been derived using *one* rounding operation, after adding some of the *original* constraints in suitable multiples. Such constraints are known as **rank-1 cuts**. In contrast constraint (1.87) was derived using constraint (1.86), which was itself derived as a rank-1 cut. Therefore constraint (1.87) is known as a **rank-2 cut**. The rank of a cut is a measure of the complexity of its derivation. For some problems (eg. The Travelling Salesman Problem) there is no limit to the rank of the facet defining constraints, giving a further indication of their computational difficulty.

As emphasised above, however, in general the derivation of Chvátal cuts is arbitrary. It is a valid 'proof' procedure for showing cuts to be valid but we have not given a systematic way of doing this. In general there is no such systematic way. If, however, we use LP to obtain the solution of the relaxed problem we can use this solution to derive some Chvátal cuts.

In the example we began by obtaining the fractional solution at C. At C constraints (1.78) and (1.79) are 'binding' i.e. they cannot be removed without altering the LP optimal solution. Also x_1 and x_2 take positive values (in LP parlance they are 'basic' variables). If we add 'slack' variables to constraints (1.78) and (1.79) to make them equations we obtain

$$x_1 + 3x_2 + u_2 \quad = \quad 15 \tag{1.89}$$

$$7x_1 - 3x_2 \quad + \quad u_3 = 23 \tag{1.90}$$

We can eliminate x_1 from equation (1.90) (using equation (1.89)) and then x_2 from equation (1.90) (using equation (1.90)) by Gaussian Elimination. Making x_1 and x_2 the subjects of their respective equations gives:

$$x_1 = \frac{19}{4} - \frac{u_2}{8} - \frac{u_3}{8} \tag{1.91}$$

$$x_2 = \frac{41}{12} - \frac{7u_2}{24} + \frac{u_3}{24} \tag{1.92}$$

Setting u_1 and u_2 ('non-basic' variables) to zero gives the fractional solution, we have already obtained, to the LP Relaxation. However we want an *integer* solution to (1.78), (1.79) and other constraints (1.79) and (1.80). Clearly since x_1 and x_2 can only take integer values this is true for u_1 and u_2 as well. (1.91) and (1.92) can be rewritten separating out their integer and fractional parts and also ensuring that the expression on the right is non-negative. This gives:

$$x_1 + u_2 + u_3 - 4 = \frac{3}{4} + \frac{7}{8}u_2 + \frac{7}{8}u_3 \tag{1.93}$$

$$x_2 + u_2 - 3 = \frac{5}{12} + \frac{17}{24}u_2 + \frac{u_3}{24} \tag{1.94}$$

Since the expressions on the right must be both *integer* and *strictly positive* we have

$$\frac{3}{4} + \frac{7}{8}u_2 + \frac{7}{8}u_3 \geq 1 \tag{1.95}$$

$$\frac{5}{12} + \frac{17}{24}u_2 + \frac{u_3}{24} \geq 1 \tag{1.96}$$

Using expressions (1.89) and (1.90) to substitute for u_2 and u_3 and eliminating fractions gives

$$7\,(15 - x_1 - 3x_2) + 7\,(23 - 7x_1 + 3x_2) \geq 2$$

i.e.

$$7x_1 \leq 33 \tag{1.97}$$

and

$$17(15 - x_1 - 3x_2) + (23 - 7x_1 + 3x_2) \geq 14$$

i.e.

$$x_1 + 2x_2 \leq 11 \tag{1.98}$$

The cuts which we have derived are known as **Gomory Cuts** and are illustrated in figure 1.8. Notice that these cuts are shallower than Chvátal cuts but they

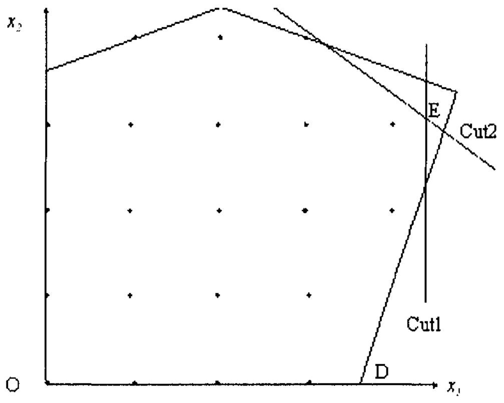

Figure 1.8. An integer programme with Gomory cuts

can be generated systematically if the Simplex Algorithm is used to solve the LP Relaxation.

We could append either or both of cuts (1.97) and (1.98) to the model and resolve. If both cuts are appended we obtain another fractional solution at E.

$$x_1 = 4\frac{11}{17}, x_2 = 3\frac{3}{7}, \text{ Objective } = 7\frac{14}{17}$$

This solution can be used to generate further cuts. Ultimately, in a finite number of steps, we will obtain the optimal integer solution at S (figure 1.2).

$$x_1 = 4, x_2 = 3, \text{ Objective } = 7$$

If, however, we had appended the Chvátal cuts (1.87) and (1.88) at the first stage we could have obtained the optimal integer solution in one step. As we have emphasised before, however, there is no known systematic way of generating Chvátal cuts.

Further discussion of the many methods of generating cutting planes is beyond the scope of this introductory chapter but can be found in eg. [19]. We should, however, mention that the generation of Gomory Cuts can be generalised to deal with the (much more common) Mixed Integer (MIP) case. This is done by Gomory [12]. Although such cuts are usually "very weak", in practice they often prove very effective.

6.2 Branch-and-Bound

This approach again starts by solving the LP Relaxation. We again use the model (1.76) to (1.80) to illustrate the method. Although this model is a PIP it will be obvious from the explanation that the method applies equally well to MIPs.

It is convenient to represent the various stages of the optimisation by a *tree.* The LP Relaxation, solved at the first stage, is represented by node 0 of the tree. In our example the solution to this is

$$x_1 = \frac{19}{4}, x_2 = \frac{41}{12}, \text{ Objective} = \frac{49}{6}$$

We then choose one of the integer variables, which has taken a fractional value, and create a branch. If we choose x_1 its possible values are illustrated in figure 1.9. There is no loss of generality in stating the following *dichotomy*

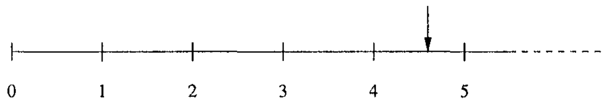

Figure 1.9. Possible values of an integer variable

$x_1 \leq 4$ **or** $x_1 \geq 5$. Whats more this rules out the current fractional solution. The choice of which integer variable to *branch-on* is heuristic. In general if a variable x takes a fractional value $N + f$ where $0 < f < 1$ then we create the dichotomy $x \leq N$ **or** $x \geq N + 1$. We append each of the alternatives extra constraints separately to the original model and represent the two new problems as two new nodes of the solution tree, illustrated in figure 1.10.

The situation is illustrated in Figure 1.11.

The solution at node 1 corresponds to the point S_1. For convenience the new nodes are numbered in order of their generation. At each node we write the solution of its LP relaxation. Notice that

(a) the objective value gets worse (*strictly* no better) as we go down a branch (smaller for a maximisation or larger for a minimisation).

(b) the LP relaxation of a new node may become infeasible.

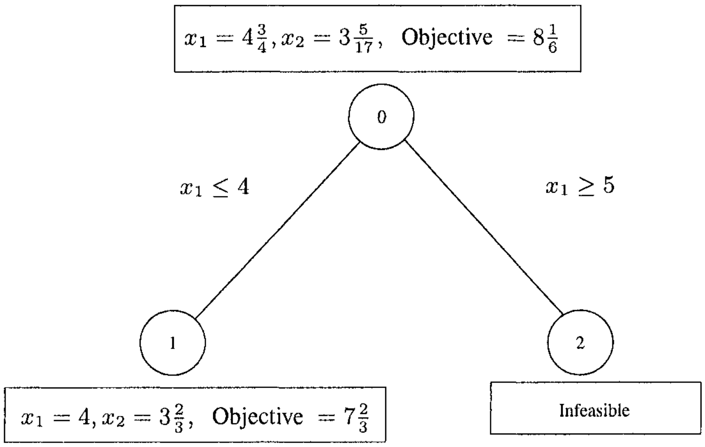
$x_1 = 4\frac{3}{4}, x_2 = 3\frac{5}{17}$, Objective $= 8\frac{1}{6}$
0
$x_1 \leq 4$
$x_1 \geq 5$
1
2
$x_1 = 4, x_2 = 3\frac{2}{3}$, Objective $= 7\frac{2}{3}$
Infeasible

Figure 1.10. The first branch of a solution tree

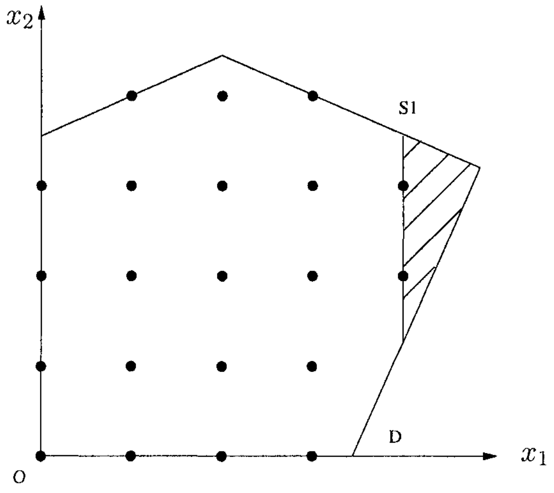

Figure 1.11. Solution space of the first branch

The process can now be continued by branching on variable x_2 to produce the solution tree in Figure 1.12.

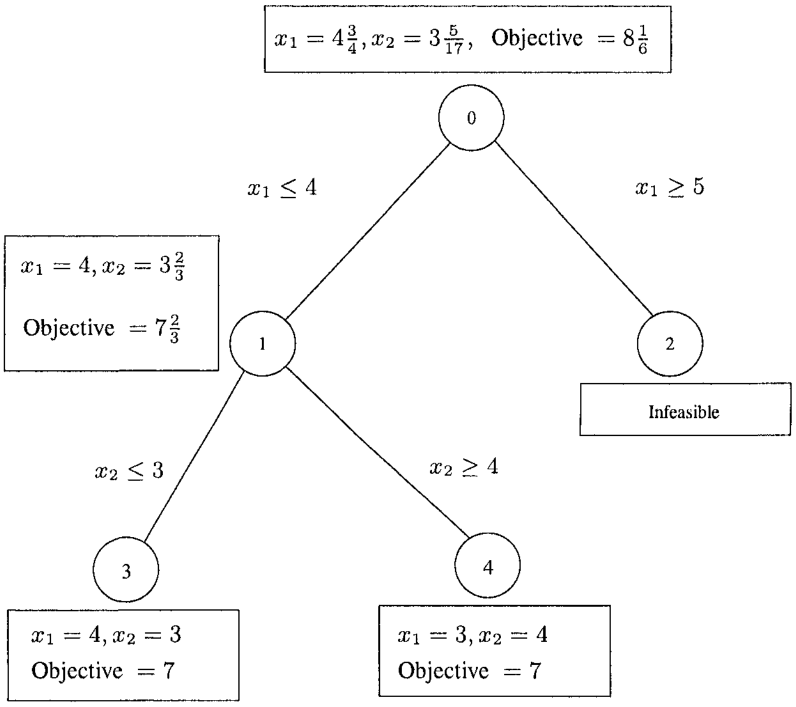

Figure 1.12. Final solution tree

In this example two alternative optimal integer solutions are produced at nodes 3 and 4.

A number of general observations need to be made about the process.

1 The method could require the generation of an exponential number of nodes. At level r we could have 2^r nodes. In practice this seldom happens. We stop going down a branch (the branch is said to be *fathomed*) for three possible reasons.

a. The LP relaxation becomes infeasible as at node 2. Therefore there can be no integer solution down this branch.

b. We obtain an integer solution as at node 3.

c. The objective value of the LP relaxation is worse than that of a known integer solution. Such a branch is said to be *bounded* because the objective value of the known integer solution provides a bound (lower bound for a maximisation or upper bound for a minimisation) on the values of the optimal integer solution. This has not happened in this case.

2 In principle this process might not terminate if the feasible region of the LP relaxation was 'open'. However, in practice, this seldom happens and can be avoided by giving the integer variables upper and lower (usually zero) bounds.

3 When a node does not lie at the bottom of a branch which has been fathomed it is known as a *Waiting Node.* Once there are no more waiting nodes (as in Figure 1.12) the best integer solution must be an optimal solution. Should no integer solution be found the model is shown to be (integer) infeasible.

4 The choices of which variables to branch on and which waiting nodes to develop are *heuristic*. How these choices are made in practice are beyond the scope of this chapter (and sometimes specific to the commercial packages used). There are, however, general strategic choices to be made regarding which waiting nodes to develop. A *depth-first* strategy involves going down individual branches of the tree in the hope of finding an integer solution quickly. Such a solution may be satisfactory (although not proven to be optimal) or used as a bound in order to fathom other future branches. A *breadth-first* strategy involves solving the LP relaxation of all waiting nodes at a particular level before proceeding to the next level. For example we solved the LP relaxation at node 2 before developing node 1 further. Which strategy is best adopted depends on the nature of the problem and whether one wants a (good) integer solution or the proven optimum. Such considerations are discussed in [24]. There are also computational considerations as to how one reoptimises the LP relaxation between nodes (eg. The *Primal* or *Dual Simplex* algorithms). These are beyond the scope of this chapter but discussed in eg. [19].

6.2.1 Other Branches. In order to illustrate the Branch-and-Bound method we branched on *integer variables* using the *dichotomy* that if an integer variable x takes the fractional value $N + f$ where $0 < f < 1$ we have

$$x \leq N \textbf{ or } x \geq N + 1$$

Other dichotomies are possible

$$\text{eg. } x = 0 \textbf{ or } x > 0$$

6.2.2 Branching on Entities. In section 1.3.2 we explained the concept of *Special Ordered Sets* and emphasised they should be regarded as *entities* rather than just as individual variables. This is to allow the concept of branching on the entity. If for example we have a S2 set $\{x_0, x_1, x_2, \ldots, x_n\}$ we require that *at most two adjacent x_j can be non-zero* (In practice we usually also stipulate the *convexity* constraint that the variables sum to 1, but this is not necessary to the concept). The branching dichotomy that we use is

$$x_0 + x_1 + \ldots + x_{j-1} = 0 \textbf{ or } x_{j+1} + x_{j+2} + \ldots + x_n = 0$$

The rational behind this is that the order of the variables is significant. Therefore, if a number of variables currently take non-zero values, we choose a variable 'in the middle' of these, x_j, and, without loss of generality, force all the variables to the left to zero or all the variables to the right to be zero. Such a branching can radically reduce the size of the tree search from branching on individual variables.

6.3 Branch-and-Cut

The use of cutting planes to solve IP models has already been discussed. There is often great merit in using them within the Branch-and-Bound method. If, for example we were to apply the Gomory Cuts (1.97) and (1.98) to the LP relaxation (at node 0) we would start with the fractional solution at E in Figure 1.8 and the Branch-and-Bound tree would take the same course as before. In general, however, one might expect it to reduce this search tree. Such cuts, incorporated into the solution process, would be known as *global cuts* since, being applied at node 0, they would be valid for the whole model.

Alternatively, or additionally, we could apply cuts at particular nodes of the tree. Such cuts would be *local cuts* since they might only be valid at that particular node and all nodes descending from it. For example, at node 1 in Figure 1.10 we have the solution represented by point S_1 in Figure 1.11. We could obtain Gomory Cuts to the LP relaxation at node 1 and apply them. One such Gomory Cut obtainable here is (1.98) (In this example it is also a global cut). Appending this cut gives the solution

$$x_1 = 4, x_2 = 3\frac{1}{2}, \text{ Objective } = 7\frac{1}{2}$$

In this, trivially small, example there is no obvious advantage in applying cutting planes in the tree search. However for larger examples it is often worthwhile.

6.4 Lagrangian Relaxation

The methods described so far have relied on the *LP Relaxation* of an IP to help in the solution process. It has already been pointed out that this is not the only possible relaxation. Indeed many relaxations are possible. One such, that has proved particularly valuable for many problems, is *Lagrangian Relaxation.* The idea here is to remove some of the constraints of the model and incorporate them in the objective function. There will then be a penalty (degradation of the objective function) in violating them but violation will not render the relaxed model infeasible. If the constraints so removed are complicated ones to satisfy then the resultant relaxed model may be easier to solve.

Because the success of the method depends on which constraints are removed and incorporated in the objective function its use is problem dependent. We illustrate the method by means of our previous small example.

$$\begin{aligned} \text{Maximise} \quad & x_1 + x_2 && (1.99)\\ \text{Subject to:} \quad & -x_1 + 2x_2 \le 7 && (1.100)\\ & x_1 + 3x_2 \le 15 && (1.101)\\ & 7x_1 - 3x_2 \le 23 && (1.102)\\ & x_1, x_2 \ge 0, \quad \text{and integer} && (1.103) \end{aligned}$$

Suppose we were to remove constraint (1.102) but penalise breaking it. The amount by which it is broken could be written as

$$V = 7x_1 - 3x_2 - 23 \qquad (1.104)$$

As we wish to minimise this quantity we could subtract $7x_1 - 3x_2$ in a suitable multiple, from the objective function. The multiple we choose can be based on the dual value of the constraint (if we start by solving the LP relaxation). The usual method is to use a technique called *subgradient optimisation* which is beyond the scope of this chapter but described in [19]. This multiplier can be regarded as a *Lagrange Multiplier.* Hence the use of the name "*Lagrangian Relaxation*" For the purposes of this illustration we will take the multiplier as 0.1. This transforms the problem to :

$$\begin{aligned} \text{Maximise} \quad & \frac{3}{10}x_1 + \frac{13}{10}x_2 && (1.105)\\ \text{Subject to:} \quad & -x_1 + 2x_2 \le 7 && (1.106)\\ & x_1 + 3x_2 \le 15 && (1.107)\\ & x_1, x_2 \ge 0, && (1.108) \end{aligned}$$

Now it might be the case, for problems with a structure, that the model above was easier to solve *as an IP* than the original model.

An example of this is if we were left with one constraint. We would then have a *Knapsack Problem* which we could solve quickly by Dynamic Programming (see Bellman [5]). Another example is with the Travelling Salesman Problem where we are left with a Spanning Tree Problem for which there exists a fast simple algorithm. It should be noted, however, that if the relaxed problem is easier to solve as an IP because it is an LP which gives an integer solution e.g. a Network Flow problem, then there is no virtue to be gained in *Lagrangian Relaxation.*

Returning to our example, solving the model above (by whatever means) yields the optimal integer solution.

$$x_1 = 3, \ x_2 = 4 \tag{1.109}$$

Note that the constraint removed (1.102) has been satisfied automatically whereas if it had been removed and given a smaller penalty it might not have been satisfied. This is clearly one of the optimal solutions to the original problem.

6.5 Heuristics

Heuristics are quick, easy-to-understand methods which are widely used to solve combinatorial problems quickly but non-optimally. They are usually problem specific. Therefore for those two reasons they are not discussed in this general chapter on *optimisation* (There is considerable discussion in Chapter 6 by Cordeau and Laporte in relation to Distribution problems). However it is very valuable to incorporate heuristics inside the Branch-and-Bound method. The overall structure of Branch-and-Bound guarantees an *optimal* solution, if carried out to completion, but the incorporation of heuristics within this structure can greatly speed the process.

We have already mentioned that heuristics can be valuable in the choices of branching variables and waiting node. They can, however, also be applied at a waiting node to try to obtain an *integer* solution before developing the node. Such an integer solution, if found, will give a *bound* on the value of the final optimal integer solution. This bound may be used, to good effect, in bounding other waiting nodes. It may also, of course, be a valuable solution in itself.

6.6 Constraint Satisfaction

This, alternative, approach to solving some Discrete Optimization problems can be very powerful. It is discussed at length in Chapter 3 by Hooker. An alternative name for this approach is Constraint Logic Programming. We outline the method here.

One of the merits of Constraint Satisfaction is that it allows a richer set of modelling expressions than simply the linear expressions of IP. The essential idea of Constraint Satisfaction is that every variable has a finite domain of possible values it may take. The predicates within this modelling language relate the variables. Therefore when a variable is assigned a particular value in its domain this may further restrict the possible values which related variables can take in their domains. For example in the *Assignment Problem* we wish to assign n objects $\{a_1, a_2, \ldots, a_n\}$ to n other objects $\{b_1, b_2, \ldots, b_n\}$ so that each object is uniquely assigned. The usual IP model would contain 0-1 variables.

$$x_{ij} = 1 \text{ iff } a_i \text{ is assigned to } b_j$$

$$= 0 \text{ otherwise}$$

For Constraint Satisfaction we would have different (less) variables

$$x_i = \text{index of object to which } a_i \text{ is assigned}$$

together with a predicate

$$\textbf{all\_different}\{x_1, x_2, \ldots, x_n\}$$

meaning that each of $a_1, a_2, \ldots, a_n$ must be assigned to a different object $b_1, b_2, \ldots, b_n$. Each x_i would initially have the domain $\{1, 2, \ldots, n\}$. When however a particular x_i is assigned to a particular value b_j (either permanently or temporarily) its index would be taken out of the domains of the other x_i (either permanently or temporarily). A tree search is performed to systematically explore the possible feasible solutions. Applying the search intelligently can result in the obtaining of a feasible solution more quickly than for IP.

As the above example demonstrates Constraint Satisfaction models are usually much more compact than IP models. The successive *domain reduction* of the variables mirrors the *Presolve* facility of many commercial IP systems as was mentioned previously.

However Constraint Satisfaction does not rely on the concept of an LP relaxation which can prove very valuable in reducing a tree search. In fact for some models (eg. the Assignment problem mentioned above) the LP relaxation yields a feasible integer solution immediately. For some, other problems with a strong *combinatorial* flavour, however, the LP relaxation is of little value.

Constraint Satisfaction sometimes is a very good way of finding feasible solutions quickly to 'needle-in-a-haystack' type problems where the finding of any feasible solution is difficult. It is less successful at finding *optimal* solutions to problems with many feasible solutions (eg. the Travelling Salesman problem). In such cases successively improved objective values have to be used to try to find solutions which are better.

There is considerable promise in designing *hybrid* systems which combine both the methods of IP and Constraint Satisfaction (see chapter 4. To date these have been most successful for specialist problems where the structure of the model can be exploited. So far there has been less progress in general hybrid systems.

6.7 Other Methods

In view of its immense practical importance, combined with the major difficulties of computation, there has been much research in alternative methods. To date these have not resulted in systems which can compete with Branch-and-Bound (augmented by Cutting Planes, Presolve etc). Logic methods have been applied to some pure 0-1 models. Here the linear constraints of IP are replaced by expressions in the *Propositional Calculus* (*Boolean Algebra*) and Logic methods (eg. *Resolution*) used to solve the resultant model. Chapter 3 by Hooker discusses some of these methods.

Another class of methods (not covered in this volume) use analogous but *discrete* concepts to those of linear algebra (Hilbert and Gröbner bases and lattice reduction, etc). A reference to such methods is Aardal, Weismantel and Wolsey [1].

The method of Projection (Fourier-Motzkin elimination) has been extended to IP models by Williams [22] and [23] and is well covered in Martin [18] with a discussion of the solution of Diophantine equations. Finally Schrijver [21] gives a very full account of the theoretical aspects of solving IP models with very many references.

References

[1] K. AARDAL, R. WEISMANTEL, AND L. A. WOLSEY. Non-Standard Approaches to Integer Programming, **Discrete Applied Mathematics**, 123 (2002), 5-74

[2] E. BALAS. Disjunctive Programming: Properties of the Convex Hull of feasible points, **Discrete Applied Mathematics**, 89 (1998), 1-46

[3] E.M.L. BEALE and J.A. TOMLIN. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables, in J. Lawrence [Ed], **Proceedings of the 5^{TH} INTERNATIONAL CONFERENCE ON OPERATIONS RESEARCH** 1969 Tavistock, London

[4] E.M.L. BEALE. Branch and Bound methods for numerical optimisation of non-convex functions, in M.M. Barritt and D. Wishart (Eds) COMPSTAT 80: **PROCEEDINGS IN COMPUTATIONAL STATISTICS 1975** pp 11-20, Physica Verlag, Wien

[5] R. BELLMAN. ***Dynamic Programming.*** 1957 Princeton University Press

[6] V. CHVÁTAL. Edmonds Polytopes and a Hierarchy of Combinatorial Problems, **Discrete Mathematics**, 4 (1973) 305-337

[7] R. J. DAKIN. A Tree Search Algorithm for Mixed Integer Programming Problems, **Computer Journal**, 8 (1965), 250-255

[8] M. R. GAREY AND D. S. JOHNSON. Computers and Interactibility: A Guide to the Theory of NP-Completeness, 1979, Freeman

[9] P.C. GILMORE AND R.E. GOMORY. A Linear Programming Approach to the Cutting Stock Problem Part I, **Operations Research**, 9 (1961) 849 – 859

[10] F. GLOVER. Improved Linear Integer Programming Formulations of Nonlinear Integer Problems, **Management Science** 224 (1975), 455-459

[11] R. E. GOMORY. Outline of an Algorithm for Integer Solutions to Linear Programs, **Bulletin of the American Mathematical Society**, 64 (1958), 275-278

[12] R. E. GOMORY. An Algorithm for the Mixed Integer Problem, Research Report, RM-2597 (1960), The Rand Corporation

[13] R. E. GOMORY. An Algorithm for Integer Solutions to Linear Programs, **Recent Advances in Mathematical Programming**, R. Graves and P. Wolf (Eds), 1983, McGraw-Hill pp. 269-302

[14] J. N. HOOKER, H. YAN, I. GROSSMANN AND R. RAMAN. Logic cuts processing networks with fixed charges, **Computers and Operations Research**, 21 (1994) 265-279

[15] R. JEROSLOW. Logic-based decision support: Mixed integer model formulation, Annals of Discrete Mathematics 40, 1989, North Holland, Amsterdam

[16] A. H. LAND AND A. G. DOIG. An Automatic Method for Solving Discrete Programming Problems, **Econometrics**, 28 (1969) 497-520

[17] E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN AND D.B. SHMOYS (Eds). **The Travelling Salesman Problem**. 1995, Wiley, Chichester

[18] R. K. MARTIN. **Large Scale Linear and Integer Optimization**, 1999, Kluwer

[19] G. L. NEMHAUSER AND L.A. WOLSEY. ***Integer and Combinatorial Optimisation.*** 1988, Wiley, New York

[20] A.J. ORMAN AND H.P. WILLIAMS. **A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem**, Working Paper LSEOR 04.67, 2004, London School of Economics.

[21] A. SCHRIJVER. **Theory of Linear and Integer Programming**, 1986, Wiley

[22] H. P. WILLIAMS. Fourier-Motzkin Elimination Extension to Integer Programming Problems, **Journal of Combinatorial Theory (A)**, 21 (1976), 118-123

[23] H. P. WILLIAMS. A Characterisation of all Feasible Solutions to an Integer Program, **Discrete Applied Mathematics**, 5 (1983) 147-155

[24] H. P. WILLIAMS. ***Model Solving in Mathematical Programming.*** Wiley, 1993

[25] H. P. WILLIAMS. ***Model Building in Mathematical Programming.*** 4th Edition, Wiley, 1999

[26] L. A. WOLSEY. Strong formulations for mixed integer programming: a survey, **Mathematical Programming,** 45 (1989) 173-191