I. Developing Linear and Integer Programming models

Chapter 1

What is modeling? Why use models?

Rather than starting with a theoretical overview of what modeling is, and why it is useful, we shall look at a problem facing a very small manufacturer, and how we might go about solving the problem. When we have gained some practical modeling experience, we shall come back to see what the benefits of modeling have been.

Section 1.1 gives a word description of the problem and its translation into a mathematical form. We then define in Section 1.2 the notion of a **Linear Program** and show how the example problem fits this form. In Section 1.3 the problem is implemented with Mosel and solved with the Xpress-Optimizer. The discussion of the results comprises a graphical representation of the solution information that may be obtained from the solver. Important issues in modeling and solving linear problems are infeasibility and unboundedness (Section 1.4). The chapter closes with reflections on the benefits of modeling and optimization (Section 1.5) and the importance of the data (Section 1.6).

1.1 The chess set problem: description

A small joinery makes two different sizes of boxwood chess sets. The small set requires 3 hours of machining on a lathe, and the large set requires 2 hours. There are four lathes with skilled operators who each work a 40 hour week, so we have 160 lathe-hours per week. The small chess set requires 1 kg of boxwood, and the large set requires 3 kg. Unfortunately, boxwood is scarce and only 200 kg per week can be obtained.

When sold, each of the large chess sets yields a profit of $20, and one of the small chess set has a profit of $5.

The problem is to decide how many sets of each kind should be made each week so as to maximize profit.

1.1.1 A first formulation

Within limits, the joinery can **vary** the number of large and small chess sets produced: there are thus two **decision variables** in our model, one decision variable per product. What we want to do is to find the **best** (*i.e.* optimal) values of these decision variables, where by 'best' we mean that we get the largest profit. We shall give these variables abbreviated names:

xs : the number of small chess sets to make

xl : the number of large chess sets to make

The number of large and small chess sets we should produce to achieve the maximum contribution to profit is determined by the optimization process. In other words, we look to the optimizer to tell us the best values of *xs*, and *xl*.

The values which *xs* and *xl* can take will always be **constrained** by some physical or technological limits. One of the main tasks in building a model is to write down in a formal manner the exact constraints that define how the system can behave. In our case we note that the joinery has a maximum of 160 hours of machine time available per week. Three hours are needed to produce each small chess set, and two hours are needed to produce each large set. So if in the week we are planning to make *xs* small chess sets and

xl large chess sets, then in total the number of hours of machine time we are planning to use is:

$$3 \cdot xs + 2 \cdot xl$$

where the $3 \cdot xs$ comes from the time making small sets, and the $2 \cdot xl$ from the time machining large sets.

Note that we have already made some assumptions here. Firstly we have assumed that the lathe-hours to machine *xs* small sets is exactly *xs* times the lathe-hours required to machine one small set. This probably will not be exactly true in practice — one tends to get faster at doing something the more one does it, so it will probably take a slightly smaller amount of time to machine the 2^{nd} and subsequent sets than the first set. But it is unlikely that this will be a very important effect in our small joinery.

The second assumption we have made is much more likely to be inaccurate. We have assumed that the time for making small **and** large sets is the sum of the times for the sets. We have not allowed for any changeover time: resetting the lathes, cleaning up, getting different size tools *etc.* In some situations, the time that we lose in changeovers can be very large compared with the time we actually spend doing constructive work and then we have to resort to more complex modeling. But for the moment, we shall assume that the changeover times are negligible.

Our first **constraint** is:

$$3 \cdot xs + 2 \cdot xl \leq 160 \quad \text{(lathe-hours)}$$

which says 'the amount of time we are planning to use must be less than or equal to the amount of time available', or equivalently 'we cannot plan to use more of the resource (time) than we have available'. The allowable combinations of small and large chess sets are restricted to those that do not exceed the lathe-hours available.

In addition, only 200 kg of boxwood is available each week. Since small sets use 1 kg for every set made, against 3 kg needed to make a large set, a second constraint is:

$$1 \cdot xs + 3 \cdot xl \leq 200 \quad \text{(kg of boxwood)}$$

where the left hand side of the inequality is the amount of boxwood we are planning to use and the right hand side is the amount available.

The joinery cannot produce a negative number of chess sets, so two further **non-negativity constraints** are:

$$xs \geq 0$$
$$xl \geq 0$$

In a similar way, we can write down an expression for the total profit. Recall that for each of the large chess sets we make and sell we get a profit of \$20, and one of the small chess set gives us a profit of \$5. Assuming that we can sell all the chess sets we make (and note that this may not always be a reasonable assumption) the total profit is the sum of the individual profits from making and selling the *xs* small sets and the *xl* large sets, *i.e.*

$$Profit = 5 \cdot xs + 20 \cdot xl$$

Profit is the **objective function**, a linear function which is to be optimized, that is, maximized. In this case it involves all of the decision variables but sometimes it involves just a subset of the decision variables. Note that *Profit* may be looked at as a **dependent variable**, since it is a function of the decision variables. In maximization problems the objective function usually represents profit, turnover, output, sales, market share, employment levels or other 'good things'. In minimization problems the objective function describes things like total costs, disruption to services due to breakdowns, or other less desirable process outcomes.

Consider some possible values for *xs*, and *xl* (see Table 1.1). The aim of the joinery is to maximize profit, but we cannot select any combination of *xs* and *xl* that uses more of any of the resources than we have available. If we do plan to use more of a resource than is available, we say that the plan **violates** the constraint, and the plan is **infeasible** if one or more constraints is violated. If no constraints are violated, the plan is **feasible**. The column labeled 'OK?' in the table tells us if the plan is feasible. Plans C and E are infeasible.

In terms of profit, plan H looks good. But is it the best plan? Is there a plan that we have not considered that gives us profit greater than 1320? To answer this question we must move to the notion of **optimization**.

Table 1.1: Values for *xs* and *xl*

	xs	*xl*	Lathe-hours	Boxwood	OK?	*Profit*	Notes
A	0	0	0	0	Yes	0	Unprofitable!
B	10	10	50	40	Yes	250	We won't get rich doing this.
C	-10	10	-10	20	No	150	Planning to make a negative number of small sets.
D	53	0	159	53	Yes	265	Uses all the lathe-hours. There is spare boxwood.
E	50	20	190	110	No	650	Uses too many lathe-hours.
F	25	30	135	115	Yes	725	There are spare lathe-hours and spare boxwood.
G	12	62	160	198	Yes	1300	Uses all the resources
H	0	66	130	198	Yes	1320	Looks good. There are spare resources.

1.2 Linear Programming

We have just built a model for the decision process that the joinery owner has to make. We have isolated the decisions he has to make (how many of each type of chess set to manufacture), and taken his objective of maximizing profit. The constraints acting on the decision variables have been analyzed. We have given names to his variables and then written down the constraints and the objective function in terms of these variable names.

At the same time as doing this we have made, explicitly or implicitly, various assumptions. The explicit assumptions that we noted were:

- For each size of chess set, manufacturing time was proportional to the number of sets made.
- There was no down-time because of changeovers between sizes of sets.
- We could sell all the chess sets we made.

But we made many implicit assumptions too. For instance, we assumed that no lathe will ever break or get jammed; that all the lathe operators will turn up for work every day; that we never find any flaws in the boxwood that lead to some being unusable or a chess set being unacceptable; that we never have to discount the sale price (and hence the per unit profit) to get an order. And so on. We have even avoided a discussion of what is the worth of a fraction of a chess set — is it a meaningless concept, or can we just carry the fraction that we have made over into next week's production?

All mathematical models necessarily contain some degree of simplification of the real world that we are attempting to describe. Some assumptions and simplifications seem eminently reasonable (for instance, that we can get the total profit by summing the contributions of the individual profits from the two sizes); others may in some circumstances be very hopeful (no changeover time lost when we swap between sizes); whilst others may just be cavalier (all the lathe operators will arrive for work the day after the World Cup finals).

Modeling is an art, not a precise science. Different modelers will make different assumptions, and come up with different models of more or less precision, and certainly of different sizes, having different numbers of decision variables. And at the same time as doing the modeling, the modeler has to be thinking about whether he will be able to *solve* the resulting model, that is find the maximum or minimum value of the objective function and the values to be given to the decision variables to achieve that value.

It turns out that many models can be cast in the form of **Linear Programming** models, and it is fortunate that Linear Programming (LP) models of very large size can be solved in reasonable time on relatively inexpensive computers. It is not the purpose of this book to discuss the algorithms that are used to solve LP problems in any depth, but it is safe to assume that problems with tens of thousands of variables and constraints can be solved with ease. So if you can produce a model of your real-world situation, without too many wild assumptions, in the form of an LP then you know you can get a solution.

So we next need to see what a Linear Programming problem consists of. To do so, we first introduce the notion of a **linear expression**. A linear expression is a sum of the following form

$$A_1 \cdot x_1 + A_2 \cdot x_2 + A_3 \cdot x_3 + \ldots + A_N \cdot x_N$$

which in mathematical notation is usually written as

$$\sum_{j=1}^{N} A_j \cdot x_j$$

where $A_1, \ldots, A_N$ are constants and $x_1, \ldots, x_N$ are decision variables. So for instance, if we have variables x, $make_P$ and $make_Q$

$$2 \cdot x - 3 \cdot make_P + 4 \cdot make_Q$$

is a linear expression, but

$$2 \cdot x \cdot make_P - 3 \cdot make_P + 4 \cdot make_Q$$

is not, as the first term contains the product of two variables.

Next, we introduce the notion of **linear inequalities** and **linear equations**. For any linear expression $\sum_{j=1}^{N} A_j \cdot x_j$ and any constant B, the inequalities

$$\sum_{j=1}^{N} A_j \cdot x_j \leq B \quad \text{and} \quad \sum_{j=1}^{N} A_j \cdot x_j \geq B$$

are **linear inequalities**, and the equation

$$\sum_{j=1}^{N} A_j \cdot x_j = B$$

is a **linear equation**. So for example our lathe-hours constraint

$$3 \cdot xs + 2 \cdot xl \leq 160$$

is a linear inequality, but

$$2 \cdot xs \cdot xl + 3 \cdot xs = 200$$

is not a linear equation because of the first term, which is a product of two decision variables.

Now, if we have decision variables

$$x_1, x_2, x_3, \ldots, x_N,$$

a linear expression

$$C_1 \cdot x_1 + C_2 \cdot x_2 + C_3 \cdot x_3 + \ldots + C_N \cdot x_N$$

and a number of linear inequalities and linear equations

$$A_{i1} \cdot x_1 + A_{i2} \cdot x_2 + A_{i3} \cdot x_3 + \ldots + A_{iN} \cdot x_N \leq B_i \quad \text{for } i = 1, \ldots, M_1$$
$$A_{i1} \cdot x_1 + A_{i2} \cdot x_2 + A_{i3} \cdot x_3 + \ldots + A_{iN} \cdot x_N = B_i \quad \text{for } i = M_1 + 1, \ldots, M_2$$
$$A_{i1} \cdot x_1 + A_{i2} \cdot x_2 + A_{i3} \cdot x_3 + \ldots + A_{iN} \cdot x_N \geq B_i \quad \text{for } i = M_2 + 1, \ldots, M_3,$$

then a **Linear Programming problem** is to

$$\begin{aligned}
\text{maximize or minimize} \quad & \sum_{i=j}^{N} C_j \cdot x_j \quad \text{(the \textbf{objective function})} \\
\text{subject to the constraints} \quad & \sum_{j=1}^{N} A_{ij} \cdot x_j \leq B_i \quad \text{for } i = 1, \ldots, M_1 \\
& \sum_{j=1}^{N} A_{ij} \cdot x_j = B_i \quad \text{for } i = M_1 + 1, \ldots, M_2 \\
& \sum_{j=1}^{N} A_{ij} \cdot x_j \geq B_i \quad \text{for } i = M_2 + 1, \ldots, M_3 \\
\text{and} \quad & x_j \geq 0 \quad \text{for each } j = 1, \ldots, N \quad \text{(\textbf{non-negativity constraints})}
\end{aligned}$$

The B_i are often called the **right hand sides** (RHS).

So, for instance, the chess set model is a linear program as it has variables xs and xl and is to

$$\begin{aligned}
\text{maximize} \quad & 5 \cdot xs + 20 \cdot xl \\
\text{subject to} \quad & 1 \cdot xs + 3 \cdot xl \leq 200 \quad \text{(kg of boxwood)} \\
& 3 \cdot xs + 2 \cdot xl \leq 160 \quad \text{(lathe-hours)} \\
& xs \geq 0 \\
& xl \geq 0
\end{aligned}$$

We have stressed the linearity condition, where the objective function and all of the constraints must be linear in the decision variables. But there are two further properties that we must have, Divisibility and Determinism.

Divisibility means that in an acceptable solution any values of the decision variables are allowed within the restrictions imposed by the linear constraints. In particular, we are not constrained to accept only whole number (integer) values for some or all of the decision variables. We have already alluded to this, when we remarked on the debate as to whether a fraction of a chess set is worth something (or, more precisely, whether a fraction *f* of a chess set is worth exactly *f* times the worth of a whole chess set).

The final requirement we have of an LP problem is that it is **deterministic** — all the coefficients in the constraints and the objective function are known exactly. Determinism is sometimes a very strong assumption, particularly if we are building planning models which extend some way into the future. Is it reasonable, for example, to assume that it will always take 7.5 days for the oil tanker to reach our refinery from the Gulf? Or that all of the lathe operators will arrive for work every day next week?

Problems where we **must** consider the variability in objective function coefficients, right hand sides or coefficients in the constraints are **Stochastic Programming** problems. We do not consider them in this book, because they are difficult to deal with, not because they are of little practical interest. In fact, it can be argued that **all** planning models have stochastic elements, and we will later demonstrate some methods for dealing with uncertainty in an LP framework.

1.3 Solving the chess set problem

Chapter 2
Typical LP model constructs

We move now to a more disciplined approach to modeling. Though we have noted that modeling is an art, this does not mean that we should treat each modeling exercise in an *ad hoc* way. Rather, we need to learn from our previous modeling experience so as to increase our productivity and skill in the next modeling work we do.

A natural way to approach any new modeling exercise is to try to understand the constraints in terms of constraint types that we have seen before and so know how to model. If we can do a disaggregation of the overall set of constraints into individual constraints that we know how to model, then we can feel quite confident that we will be able to complete the whole model.

So we shall look at some typical constraint types. We will first give various 'word descriptions' of constraints, that is, phrases in which the constraint might be articulated by the person responsible for the system we are modeling. We will then give an algebraic formulation of the constraint, and see how it might be generalized. The presentation below is given in terms of the following constraint types: bounds (Section 2.1), flow constraints (Section 2.2), simple resource constraints (Section 2.3), material balance constraints (Section 2.4), quality requirements (Section 2.5), accounting constraints (Section 2.6), blending constraints (Section 2.7), modes (Section 2.8), soft constraints (Section 2.9), and as a special form of non-constrained 'constraints', objective functions (Section 2.10),

It should be noted that these constraint types are somewhat arbitrary — another presentation could have categorized them in different ways.

2.1 Simple upper and lower bounds

A) 'Marketing tells me that we cannot sell more than 100 units of Product 4 this period.'

B) 'I'm committed to sending at least 20 tonnes of Blend 6 to Boston tomorrow.'

C) 'I have to send exactly 30 tonnes of Blend 5 to Paris next period.'

D) 'Electricity can flow in either direction along the Channel Cable.'

In **A**, one of the decision variables will be the amount of Product 4 that we sell, and suppose this is variable $sell_4$. The words say that there is an **upper limit** or **upper bound** of 100 on the single variable $sell_4$, *i.e.*

$$sell_4 \leq 100$$

In case **B**, we have a restriction in the opposite direction. If the decision variable is, say, $send_{b6,Boston}$, then we have a **lower limit** or **lower bound** on the single variable,

$$send_{b6,Boston} \geq 20$$

Note that this is more restrictive than the normal non-negativity constraint

$$send_{b6,Boston} \geq 0$$

We can see case **C**, where the decision variable might be called $send_{b5,Paris}$, as an example of a variable being **fixed**, or equivalently that its lower and upper bounds are identical, *i.e.*

$$send_{b5,Paris} = 30$$

or

$$send_{b5,Paris} \geq 30 \text{ and } send_{b5,Paris} \leq 30$$

One might ask 'why have $send_{b5,Paris}$ as a decision variable at all, as it is fixed and there is no decision to make about its value?'. If every day we always sent exactly 30 tonnes to Paris then it might be legitimate to remove this from the decision process, remembering to account for the resources that are used in the blend, the transport capacity that is used up in moving the blend, *etc.* But of course there will inevitably come a day when Paris needs more of Blend 5, and we have to spend time recalculating the resource availabilities, transport capacity *etc.*, net of the new Parisian demand. We will get things wrong unless someone remembers to do the recalculation; and of course forgetting to redo the sums is likely when we are having to rejig our production plan because of the extra demands from other cities in France (it is Bastille Day tomorrow). What we gain by removing one variable from the model is not worth the risk arising from later inflexibility.

The last case, **D**, expresses the fact that sometimes a variable is not bound by the normal non-negativity constraint, and in this situation it is said to be **free**. Since, as we have mentioned before, all decision variables are by default assumed to be non-negative, we have to explicitly tell our modeling software that the decision variable here (which we shall call *eflow*), denoting the flow from Britain to France where a positive value means that electricity is flowing from Britain to France and a negative value means that it is flowing in the reverse direction is not non-negative

$$-\infty \leq eflow \leq +\infty$$

In Mosel, this is indicated by the notation

```
eflow is_free
```

Free variables can be replaced by normal non-negative variables, *e.g.*

$$eflow = eflow_{BF} - eflow_{FB}$$

where $eflow_{BF}$ ($eflow_{FB}$) is the non-negative flow from Britain to France (respectively, France to Britain), but since optimization software can handle free variables directly there may be no reason to do this transformation. If, however, there is some asymmetry in the flows (perhaps a flow from Britain to France is taxed in some way by the French government, whereas a flow in the other direction is not taxed), the $eflow_{BF}$ and $eflow_{FB}$ variables may be required anyway by other parts of the model, so the substitution is harmless, or even necessary.

2.2 Flow constraints

Flow constraints arise where one has some sort of divisible item like electricity, water, chemical fluids, traffic *etc.* which can be divided into several different streams, or alternatively streams can come together. Some word formulations might be

A) 'I have got a tank with 1000 liters in it and 3 customers C1, C2 and C3 to supply.'

B) 'I buy in disk drives from 3 suppliers, S1, S2 and S3. Next month I want to have at least 5000 disk drives arriving in total.'

C) 'I have 2 water supplies into my factory, S1 and S2. I get charged by the water supplier for the amount of water that enters my site. I lose 1% of the water coming in from S1 by leakage and 2% of that from S2. My factory needs 100,000 gallons of water a day.'

Case **A** is where there is an outflow from a single source, whilst case **B** is where we want to model a total inflow. Case **C** is a little more complicated, as we have to model losses.

In case **A** we have flow variables $supply_1$, $supply_2$ and $supply_3$, being the amounts we supply to the three customers. The total amount supplied is the sum of these three decision variables *i.e.* $supply_1 + supply_2 + supply_3$, and this total must be less than or equal to the 1000 liters we have available. Thus we have

$$supply_1 + supply_2 + supply_3 \leq 1000$$

(providing we have defined the units of supply to be liters).

Case **B** is very similar. The obvious decision variables are buy_1, buy_2 and buy_3, and the total amount we buy is $buy_1 + buy_2 + buy_3$. Our requirement is for at least 5000 drives in total, so the constraint is

$$buy_1 + buy_2 + buy_3 \geq 5000$$

If we wanted exactly 5000 drives, then the constraint would be

$$buy_1 + buy_2 + buy_3 = 5000$$

Case **C** is more challenging. 'I have 2 water supplies into my factory, S1 and S2. I get charged by the water supplier for the amount of water that enters my site. I lose 1% of the water coming in from S1 by leakage and 2% of that from S2. My factory needs 100,000 gallons of water a day.'

There are several different ways we might approach the modeling. One way is to define decision variables as follows: buy_1 and buy_2 are the amounts we buy from supplier 1 and 2 respectively, and get_1 and get_2 are the amounts that we actually get (after the losses). If we measure the decision variables in thousands of gallons, then we have the requirement constraint

$$get_1 + get_2 = 100$$

and we can model the losses as follows (**loss equations**):

$$get_1 = 0.99 \cdot buy_1$$
$$get_2 = 0.98 \cdot buy_2$$

since, for example, we only receive 98% of the water from supplier 2 that we have to pay for. Then the objective function would, among other entries, have terms $PRICE_1 \cdot buy_1 + PRICE_2 \cdot buy_2$, where $PRICE_1$ and $PRICE_2$ are the prices per thousand gallons of water from the two suppliers.

We have the option of using the loss equations to substitute for buy_1 and buy_2 (or, alternatively, get_1 and get_2), and doing so would reduce the number of equations and variables by 2. But as we have discussed before, it almost certainly is not worth the effort, and the LP solver should be clever enough to do the work for us automatically anyway if it thinks it will reduce the solution time.

2.3 Simple resource constraints

We have already seen examples of this sort of constraint in the chess set problem where, to remind you, we had limited amounts of lathe-hours and boxwood, and we had to create a production plan that used neither too many lathe-hours nor too much boxwood.

If we generalize what we mean by a resource, all linear programs can be formulated as maximizing some objective function subject to not using any more of any resource than we have available. But it is not particularly helpful to think of everything as a resource, so we will give more word formulations here which involve real resources.

A) 'I can only get hold of 10,000 connectors a month. Each Industry PC I make needs 8 connectors and each Home PC I make requires 5 connectors.'

B) 'Each tonne of chemical C1 uses 50 grams of a rare catalyst and 1 kg of a certain fine chemical, whereas each tonne of chemical C2 requires 130 grams of the catalyst and 1.5 kg of the fine chemical. I can only afford to buy 10 kilograms of catalyst per month and I can only acquire 200 kg of the fine chemical each month.'

C) 'If I do one unit of activity i it takes up $COST_i$ of my disposable income. My disposable income next week is $150.'

If we consider just one scarce resource then a resource constraint can be characterized by the activities that potentially use up some of that resource, the level at which we are planning to conduct those activities, the availability of the resource and, finally, the amount of the resource that is used per unit of each activity.

These latter numbers, the resources used per unit of activity, are called **technological coefficients**. because they are determined by the technology that we are using. Typically, when we have several products which are each made from several resources, we will have a **technological coefficient matrix** which gives the number of units of each resource required to make one unit of the product. It would be presented in the form of Table 2.1, where, for example, the entry 0.8528 means that to make 1 unit of Product 3 we require 0.8528 units of Resource 3. We have to be careful to use consistent and sensible units for how we measure products and resources.

Case **A** where we stated that 'I can only get hold of 10,000 connectors a month. Each Industry PC I make needs 8 connectors and each Home PC I make requires 5 connectors' has a simple technological matrix:

If we have decision variables $make_I$ and $make_H$, being the number of Industrial and Home computers we plan to make, then the number of connectors we plan to use is the number used making industrial computers plus the number used making Home computers. If we make 1 Industrial computer then we use 8 connectors, so if we make $make_I$ Industrial computers we use $8 \cdot make_I$ connectors; and if we make 1 Home computer then we use 5 connectors, so if we make $make_H$ Home computers we use $5 \cdot make_H$

Table 2.1: Technological coefficient matrix

	Resource 1	Resource 2	Resource 3	...	Resource R
Product 1	1.28235				0.24
Product 2	0.95474				1.8361
Product 3	12.33312	6.128	0.8528		
...				*etc.*	
Product P	11.2322				5.208

Table 2.2: Simple technological coefficient matrix

	Connector
Industry PC	8
Home PC	5

connectors. So in total if we make $make_I$ Industrial computers and $make_H$ Home computers we will use $8 \cdot make_I + 5 \cdot make_H$ connectors. This total must not be greater than the number available *i.e.*

$$8 \cdot make_I + 5 \cdot make_H \leq 10000$$

In case **B** we said 'Each tonne of chemical C1 uses 50 grams of a rare catalyst and 1 kg of a certain fine chemical, whereas each tonne of chemical C2 requires 130 grams of the catalyst and 1.5 kg of the fine chemical. I can only afford to buy 10 kilograms of catalyst per month and I can only acquire 200 kg of the fine chemical each month.' So we have two products and two resources.

Table 2.3: Resource use

	Catalyst	Fine Chemical
C1	50	1.0
C2	130	1.5

We measure the catalyst in grams, and the fine chemical in kg. Defining decision variable $make_1$ and $make_2$ to be the number of tonnes of the two chemicals (C1 and C2) we plan to make, and using an argument similar to the one in case A), we get two constraints, one for the catalyst and one for the fine chemical.

$$50 \cdot make_1 + 130 \cdot make_2 \leq 1000 \quad \text{(catalyst)}$$
$$1.0 \cdot make_1 + 1.5 \cdot make_2 \leq 200 \quad \text{(fine chemical)}$$

We have been careful to keep the units of measurement consistent.

In general, we have one constraint for each resource, relating the amount we are planning to use to the amount available.

Case **C** is now hardly challenging. 'If I do one unit of activity i it takes up $COST_i$ of my disposable income. My disposable income next week is \$150.' The decision variables are $doact_i$, the amount of activity i that I plan to do. Making the linearity assumption that if I do $doact_i$ units of activity i it costs me $COST_i \cdot doact_i$, my total planned expenditure is $\sum_i COST_i \cdot doact_i$ and this can be no more than the money I have (assuming credit is not an option for me!). Then the constraint is

$$\sum_i COST_i \cdot doact_i \leq 150$$

2.4 Material balance constraints

Many systems have constraints which can be expressed in words as 'what goes out must in total equal what comes in'. In other words, there can be no loss of mass when some particular process takes place. If the process is some sort of material flow and the thing that is flowing is incompressible, then sometimes the constraint is expressed in terms of conservation of volume.

Pictorially if we have a process with inflow variables $flowin_i$ where i ranges from 1 to N_{IN} and outflows $flowout_j$ where j ranges from 1 to N_{OUT}, and an inflow L from the outside world, we might have

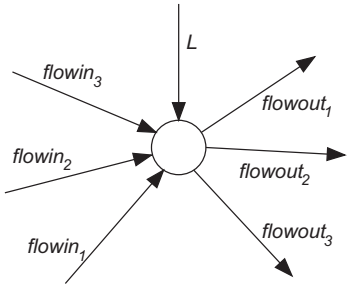

Figure 2.1: Material flow balance in a node

If the flow variables are measured in terms of mass or weight then we will have

$$L + \sum_{i=1}^{M_{IN}} flowin_i = \sum_{j=1}^{N_{OUT}} flowout_j$$

A particular form of material balance constraint occurs when we are accounting for the flow of materials between time periods in a **multi-time period model**, that is, a model where we represent time as a series of intervals, not concerning ourselves with the details of when during an interval events happen. Suppose that we have a simple factory that only makes one product and that we are trying to decide the manufacturing levels ($make_t$) for the next NT periods, *i.e.* t ranges from 1 to NT. Another set of decision variables are $sell_t$, the amount of the product that we decide to sell in time period t. If we have the possibility of storing product from one time period to the next time period we can introduce a further set of decision variables $store_t$ which are the amounts of product to have in stock (inventory) at the end of time period t.

Let us consider the material balance in time period t. In words we can say 'the stock at the end of time period t is equal to the stock at the beginning of time period t, plus what we make, minus what we sell.'

We are faced with a slight problem now — we do not know the decision variable for the stock at the beginning of time period t but we can see that, assuming there is no loss of stock, the stock at the beginning of time period t is the same at the end of time period t -1. So in different words, the previous statement can be phrased as 'the stock at the end of time period t is equal to the stock at the end of time period $t - 1$, plus what we make in period t, minus what we sell in period t'. In algebraic form this can be written

$$stock_t = stock_{t-1} + make_t - sell_t$$

This is deceptively simple but it contains one flaw: there is no time period preceding time period 1, so there is no $stock_0$ variable. We need a special constraint that relates to time period 1 only, and says in words 'the stock at the end of time period 1 is equal to the opening inventory $SINIT$, plus what we make in time period 1, minus what we sell in time period 1', or in algebra

$$stock_1 = SINIT + make_1 - sell_1$$

Other material balance constraints arise when we are talking about raw materials that we buy in from an outside supplier and use in our production process. A word description might be, 'for a particular raw material, the stock of the raw material at the end of time period t is equal to the stock at the end of time period $t - 1$, plus the amount of the raw material that we buy in time period t, minus the amount of raw material that we use in making the products during time period t'. Then considering one raw material we might have decision variables $rbuy_t$, $rstock_t$ and $ruse_t$, so the constraints for all but the first time period are

$$rstock_t = rstock_{t-1} + rbuy_t - ruse_t$$

We have to remember again that there is a special constraint for time period 1 as we already know the opening stock level in that period: it is what we have in stock right now.

It is also likely that the $ruse_t$ variables will be related to the decision variables in another part of the model which represent how much product we are going to make. If we have products indexed by $p = 1, \ldots, NP$ with technological coefficients REQ_{pr}, the number of units of raw material r required by 1 unit of product p, and decision variables $make_{pt}$ denoting the amount of product p that we make in time period t, we can immediately write down an expression for the amount of raw material we use in time period t ($ruse_t$)

as

$$ruse_t = \sum_{p=1}^{NP} REQ_{pr} \cdot make_{pt}$$

(see Section 2.3 'Simple resource constraints' above for an explanation of this).

Since we have just written down an equation for $ruse_t$ we might consider substituting for it in the constraint in which it occurs. This would generally not be a particularly good thing to do, but not particularly bad either. It would reduce the number of variables a little but we are probably going to have to use these variables somewhere else in the model anyway and so the substitution would have to take place everywhere the $ruse_t$ variables appeared in the model. The model would certainly be less comprehensible, and consequently harder to maintain.

A new form of material balance equations of the multi-period type occurs where we have **fixed demands** for our product or products in the *NT* time periods. In other words, the selling decision variables ($sell_t$ in the example above) are fixed. We have the choice of expressing this as a set of simple equality constraints with $sell_t$ variables *e.g.*

$$sell_t = MUSTSELL_t$$

or by substituting the $sell_t$ variables by $MUSTSELL_t$ everywhere they occur. This time the decision as to whether to substitute for the $sell_t$ variables is probably a matter of choice — some modelers will eliminate variables while others might put in the simple equality constraint arguing that the $sell_t$ variables will probably at some stage in the future be of interest in themselves.

2.5 Quality requirements

Here are some word statements

A) 'I cannot have more than 0.02% of sulphur in this gasoline.'

B) 'There must be at least 3% of protein in this dried apricot mixture.'

C) 'This foodstuff must have no less than 5% fat but no more than 10%.'

These sorts of quality requirements frequently occur when we are blending together various raw materials with differing properties to create a final product. Typically each of the raw materials will come with a set of properties, for instance in foodstuffs it might be the fat content or the percentage of carbohydrates; in the petrochemical industry it might be sulphur content or the octane number. For some characteristics of the final product we require that quality specifications are adhered to. For 'bad' things these are usually expressed as a maximum percentages; for desirable properties they will be expressed as a minimum percentage and other properties may have to lie between specified lower and upper bounds (perhaps so that the product has stability or other characteristics that we desire). An extreme form of the latter case is where we have to have an exact percentage of a particular characteristic in the final product.

Consider a number *NR* of raw materials and just one characteristic that we are concerned about in the final product. Suppose that one unit of raw material r contains $CONT_r$ units of the characteristic (for instance $CONT_3$ might be 0.1 if raw material number 3 is 10% fat and the characteristic we are considering is fat). As usual with LP we assume that we know the *CONT* data precisely.

In many manufacturing processes it is reasonable to assume that the characteristic in the resulting end product comes from blending the raw materials linearly. For example if we blend together one kg of a gasoline with 0.01% sulphur and 1 kg of gasoline with 0.03% sulphur, then we will get 2 kg of the mixture with 0.02% sulphur. Whether this assumption of linearity is correct very much depends upon the physics or chemistry of the blending process and in some industries it certainly does not hold. However, making the assumption of linearity, we can see that the proportion of the characteristic in the final product is given by

$$p = proportion = \frac{\text{total mass of constituents}}{\text{total mass of blend}}$$

If the decision variables are $ruse_r$ for the amount of raw material r to use, then we have

$$p = \frac{\sum_{r=1}^{NR} C_r \cdot ruse_r}{\sum_{r=1}^{NR} ruse_r}$$

and we can apply the desired inequality (or equality) to this proportion. For instance, $p \leq 0.1$ (*i.e.* 10%) becomes

$$\frac{\sum_{r=1}^{NR} C_r \cdot ruse_r}{\sum_{r=1}^{NR} ruse_r} \leq 0.1$$

At first sight this does not seem like a linear inequality and indeed it is not as it is a constraint applying to the ratio of two linear expressions. However, by cross multiplying we can get the constraint into the form

$$\sum_{r=1}^{NR} C_r \cdot ruse_r \leq 0.1 \sum_{r=1}^{NR} ruse_r$$

Note: we have to be careful when multiplying inequalities. The direction of the inequality is preserved if we multiply by a non-negative amount, but is reversed if we multiply by a negative amount. Here, we are multiplying both sides of the inequality by the sum of a set of non-negative variables, so the direction of the inequality is maintained.

If we collect together two terms for each decision variable $ruse_r$ we get the linear inequality

$$\sum_{r=1}^{NR} (C_r - 0.1) \cdot ruse_r \leq 0$$

2.6 Accounting constraints, non-constraining 'constraints'

A typical way of building up a model is to construct it in quasi-independent pieces. So, for instance, we might have a constraint expressed as 'the total cost of operating region 1 plus the cost of operating region 2,... plus the cost of operating region 16 must be less than $10M', and then later on in the model we have 16 different equations which capture the cost of operating each of the regions. If we had decision variables *totalcost* and $rcost_i$ this would be expressed as

$$totalcost = \sum_{i=1}^{16} rcost_i$$

where $rcost_i$ = $some\_expression_i$ for i = 1 to 16.

Obviously it is possible to eliminate the $rcost_i$ variables (the cost of operating region *i*) by substituting $some\_expression_i$ into the equation defining *totalcost* but it is very likely that we will want the costs of operating different regions to appear in some report, so we might as well let the LP system calculate the values of these variables for us. We call the $rcost_i$ variables **accounting variables** and the constraints that define them **accounting constraints**.

Whether you choose to use the LP system to do the accounting for you, or do them afterwards when producing reports, is a matter of taste. In the past, when LP systems were limited and one had to squeeze the problem into as few constraints as possible, it was obviously a good idea to eliminate accounting variables and then to calculate their values from the optimum values of the variables contributing to them. Now, where a few hundred extra constraints are of no consequence it is better to leave accounting variables and constraints in the model.

There is however one subtlety that we must be aware of: it is possible that an accounting variable does not necessarily have to be non-negative. Suppose we had variables $revenue_i$ and $expenditure_i$ which were the revenue and expenditure of plant *i*. If we were to define accounting variables $profit_i$ and accounting constraints

$$profit_i = revenue_i - expend_i$$

to pick up the values of the $profit_i$ variables then we must remember that it is possible for a plant to have negative profitability and so the $profit_i$ variables **must** be declared as **free variables** (unless for some reason we have to run each plant in a profitable mode).

Forgetting that decision variables are by default non-negative is a common cause of error with accounting variables. However, if one just wants the LP system to do accounting for you, it is possible to add extra unconstrained linear expressions to a model, letting the LP system work out the value of the linear expression for you. It used to be the case that adding lots of these expressions could slow down the LP solution process, but modern solvers recognize such constraints and discard them during the solution process, only reinstating them when the rest of the LP has been solved.

2.7 Blending constraints

A very common use of LP modeling is to deal with the situation where we have a set of inputs each with certain percentages of different characteristics and we want to blend these inputs together to get a final

product with certain desired characteristics. Blending typically falls into one of two major categories: either we use **fixed recipes** where the proportion of different inputs is determined in advance (rather like the recipes you see in cookery books which specify the exact weights of the constitutes required to make a certain weight of cake) or we have **variable recipes** where it is up to us to decide, perhaps within limits, the proportions of the inputs that we are going to blend together.

An example of a variable recipe might be in the manufacture of animal feed stuffs which can be made from a wide variety of different raw materials, not necessarily always mixed in the same proportions. An interesting example of recipes which pretend to be fixed but in fact are variable are those found in books that tell you how to mix cocktails. They usually pretend that the proportions of gin, tequila, vermouth etc are fixed, but of course anyone who has been in the position of having restricted stocks of any of these raw materials knows that within limits the recipes are very variable.

Suppose we have 3 raw materials and the constraint in words is 'the ratios of raw material 1, raw material 2 and raw material 3 are 6:3:1'. If we have decision variables raw_1, raw_2 and raw_3 representing the weight of the materials in the blend, then the total weight in the blend is $raw_1 + raw_2 + raw_3$ and we have the constraints

$$\frac{raw_1}{raw_1 + raw_2 + raw_3} = \frac{6}{10}$$
$$\frac{raw_2}{raw_1 + raw_2 + raw_3} = \frac{3}{10}$$
$$\frac{raw_3}{raw_1 + raw_2 + raw_3} = \frac{1}{10}$$

These are ratio constraints, not linear equations, but as before we can convert them into linear equations by cross multiplying to get

$$(1 - 0.6) \cdot raw_1 = 0.6 \cdot raw_2 + 0.6 \cdot raw_3, \quad \text{i.e.}$$
$$0.4 \cdot raw_1 = 0.6 \cdot raw_2 + 0.6 \cdot raw_3$$

The cross multiplication preserves the direction of the inequality since the denominator is always non-negative.

Similarly we get

$$0.7 \cdot raw_2 = 0.3 \cdot raw_1 + 0.3 \cdot raw_3, \quad \text{and}$$
$$0.9 \cdot raw_3 = 0.1 \cdot raw_1 + 0.1 \cdot raw_2$$

One of these equations is redundant as it is implied by the other two but it does no harm to put all three equations into the model. In fact it is probably a good idea to put all three equations down as inevitably at some time in the future we will have a fourth raw material and if we try to be too clever in eliminating redundant constraints we will forget that we have omitted a previously redundant equation, and make a mistake that will be hard to detect.

2.8 Modes

A generalization of blending constraints is where we have *M* inputs and *N* outputs which must be in strict proportions. As an example, this might be expressed as 'My 3 inputs have to be used in the ratio of 2 : 1.6 : 2.2 and they produce outputs of 1.6 of output 1 and 1.7 of output 2'. So, for example we might put in 1 kg of input 1, 0.8 kg of input 2, and 1.1 kg of input 3, and get out 0.8 kg of output 1 and 0.85 kg of output 2.

It is easy to see that if we have just one output then this is a simple fixed ratio blending example.

Such *M*-input/*N*-output constraints often arise where we have a plant that we can operate in different ways (**modes**), and the ratios differ for different modes. At any point in time, the plant can only be in one mode.

Consider a very simple example, where we have 3 inputs, 2 outputs and 3 possible operating modes. We present the operating characteristics in the tables below, where we have defined a unit of a mode as 1 hour *i.e.* we have shown the kg of each input used, and output produced, by the plant.

The decision variables are the number of hours the plant spends in each mode *m*, say $usemode_m$. Then the usage of Input 1, for instance, is $20 \cdot usemode_1 + 22 \cdot usemode_2 + 24 \cdot usemode_3$ and the production of output 2 is $17 \cdot usemode_1 + 20 \cdot usemode_2 + 24 \cdot usemode_3$.

Table 2.4: Fixed ratio blending example

	Quantities used/produced		
	Mode 1	Mode 2	Mode 3
Input 1	20	22	24
Input 2	16	14	13
Input 3	22	21	18
Output 1	16	15	14
Output 2	17	20	24

In any given planning period we will also have a constraint on the total number of hours the plant spends in all the modes. For instance, if we are doing a weekly plan (168 hours), then we will have the constraint that

$$usemode_1 + usemode_2 + usemode_3 \leq 168$$

We are assuming that we can work at most 168 hours in the week, and that we do not lose any time changing from one mode to another. The latter assumption may be totally unrealistic for some plants, and if we cannot assume that change-over times are negligible then we have to use an Integer Programming formulation.

2.9 Soft constraints and 'panic variables'

The constraint that we have just modeled, that we can run a plant for at most 168 hours, is an example of a **hard constraint**. It is impossible to get more than 168 hours into a week. Other examples of hard constraints are ones that relate to physical, chemical or engineering properties, such as a boiler's capacity, or a reaction rate. Other hard constraints come from accounting or definitional constraints (for instance, profit = revenue-cost is a hard constraint, as it is just a definition really).

But there are other, more 'economic' constraints where in reality if we have to violate a constraint we can do so at a cost. If there is a scarce resource, the economic environment invariably sets up a market in that resource, and we can go to the market if we really do need to acquire more of that resource. (This is somewhat of a generalization as a particular resource may actually be so specific to us that we are the only people who can make it to the standards or in the locality or to the timescale we require.)

Consider again the statement of case B of Section 2.3 'Simple resource constraints' above: We said 'Each tonne of chemical C1 uses 50 grams of a rare catalyst and 1 kg of a certain fine chemical, whereas each tonne of chemical C2 requires 130 grams of the catalyst and 1.5 kg of the fine chemical. I can only afford to buy 10 kilograms of catalyst per month and I can only acquire 200 kg of the fine chemical each month.' The key part of this phrasing is the last sentence, where we say we can only afford so much of the catalyst, and we can only acquire 200 kg of the fine chemical. We expressed the fine chemical constraint as

$$1.0 \cdot make_1 + 1.5 \cdot make_2 \leq 200 \quad \text{(fine chemical)}$$

i.e. as a hard constraint. But suppose that if we were willing to pay P per kg we could get more of the fine chemical. Then we could introduce an extra decision variable *buyFC* to represent the number of kg of fine chemical we bought at this price. If we were building a profit maximizing LP then we would have to add a term $-P \cdot buyFC$ to the objective function, accounting for the money we pay for the fine chemical, but now the constraint becomes

$$1.0 \cdot make_1 + 1.5 \cdot make_2 \leq 200 + buyFC \quad \text{(fine chemical, with buy-in)}$$

We now have a **soft constraint**, *i.e.* we can violate the original constraint but at a penalty to our objective function. The *buyFC* variable can be considered a **panic variable**: it is a variable which is there in case we cannot manage with our normal resources and have to resort to the market to satisfy our needs.

The way we have introduced panic variables is not the way they are most commonly used in modeling. We have mentioned before that getting an infeasible solution to an LP is very difficult to diagnose, and good modelers have learned the art of what we might call 'defensive modeling'. One of the worst things that can happen to a model when it has been deployed to end-users is that it gives an infeasible solution message to the poor end-user, who then does not know what to do. Defensive modelers try to avoid this by adding panic variables to constraints that might possibly be violated, at the same time trying to get estimates of how much it will realistically cost to violate the constraint. Often the end-user can help in this, as they will know what can be done to get extra resources — for instance, if it is a question of shutting down the plant or renting a plane to fly in the scarce resource, you may well decide to rent the plane.

Panic variables do not just apply to resource availability constraints; they can also be added to demand constraints. For instance, we might have a constraint that a particular customer *C* has to have a total supply from our depots of 100 tonnes, modeled as

$$\sum_{d \in DEPOTS} flow_{dC} \geq 100$$

If, however, it is just not possible to supply him then we may undersupply. There may be a contractual cost to doing this, in which case we know what the direct cost will be, or we might have some measure of the cost in terms of loss of goodwill (admittedly these costs are hard to estimate, and can be very subjective).

If a panic variable is non-zero in an optimal solution, then we can infer one of two things: either we have put in too low a penalty cost, or we are truly being forced to use the panic variable to achieve feasibility. Rerunning the model with higher penalty costs will enable us to distinguish these cases. But in either case we have an implementable solution, and we are not left with an end-user staring at a computer screen wondering what on earth to do.

2.10 Objective functions

An objective function is not a constraint, but since it consists of a linear expression usually involving lots of variables its construction has many similarities to the process of building constraints. In fact, now we have so extensively covered modeling constraints there is little new to say about modeling an objective function.

The objective is usually cost (when we minimize) or profit (when we maximize). Most LP systems minimize by default and the industry MPS standard for presenting an LP problem to an optimizer does not specify whether the objective is to be minimized or maximized, so everyone has at one time or another selected the wrong sense for optimization, and come up with a silly answer. One feels foolish, but it is usually obvious that a mistake has been made.

Problems do exist where there is no objective function, and the LP optimizer is being used just to obtain a feasible solution, or to answer the question in a design study as to whether the constraints are too restrictive. But such problems are rare.

2.10.1 Minimax objective functions

Two apparently non-linear objective functions can be converted into LP forms by tricks. The first is where we wish to:

$$\text{minimize } \max(t_1, t_2, t_3, \ldots, t_N)$$

i.e. minimize the maximum of a set of decision variables. We can model this by introducing a new decision variable, say *s*, and then

$$\begin{aligned} \text{minimize} \quad & s \\ \text{subject to} \quad & s \geq t_1 \\ & s \geq t_2 \\ & s \geq t_3 \\ & \ldots \\ & s \geq t_N \end{aligned}$$

You can see that *s* has to be no less than each of the *t*'s, and the minimization objective will force it down to take the value of the largest *t*. Unfortunately, the same trick cannot be applied where we want to minimize $\min(t_1, t_2, t_3, \ldots, t_N)$ or maximize $\max(t_1, t_2, t_3, \ldots, t_N)$.

2.10.2 Ratio objective functions

The second trick enables us to deal with a **ratio objective function** of the form

$$\begin{aligned} \text{minimize} \quad & Obj = \frac{\sum_j N_j \cdot x_j}{\sum_j D_j \cdot x_j} \\ \text{subject to LP constraints} \quad & \sum_j A_{ij} \cdot x_j \sim R_i \end{aligned}$$

where $\sim$ is one of $\leq$, $\geq$ or =.

Define new variables $d = \frac{1}{\sum_j D_j \cdot x_j}$ and $y_j = x_j \cdot d$ (*i.e.* $x_j = y_j \ / \ d$). Then the objective becomes

$$\begin{aligned} Obj &= d \cdot \textstyle\sum_j N_j \cdot x_j \\ &= \textstyle\sum_j N_j \cdot d \cdot x_j \\ &= \textstyle\sum_j N_j \cdot y_j \end{aligned}$$

The definition of d can be linearized by cross multiplying:

$$1 = \sum_j D_j \cdot d \cdot x_j = \sum_j D_j \cdot y_j$$

and into the normal LP constraints we can substitute for x_j thus:

$$\sum_j A_{ij} \cdot x_j \sim R_i$$

or (multiplied by d)

$$d \cdot \sum_j A_{ij} \cdot x_j \sim d \cdot R_i$$

or

$$\sum_j A_{ij} \cdot d \cdot x_j \sim d \cdot R_i$$

i.e. or

$$\sum_j A_{ij} \cdot y_j \sim d \cdot R_i$$

The resulting LP has variables d and y_j. When it has been solved, the optimal values of the original x_j variables can be recovered from the definition $x_j = y_j \ / \ d$.

Note that we are in trouble if variable d can become 0. We also have to taken care if d is always negative, as the signs of the inequalities must be reversed when we multiply by d above.

Chapter 3
Integer Programming models

Though many systems can accurately be modeled as Linear Programs, there are situations where discontinuities are at the very core of the decision making problem. There are three major areas where non-linear facilities are required

- where entities must inherently be selected from a discrete set of possibilities;
- in modeling logical conditions; and
- in finding the global optimum over functions.

Modeling languages let you model these non-linearities and discontinuities using a range of discrete entities and then a Mixed Integer Programming (MIP) optimizer can be used to find the overall (global) optimum of the problem. Usually the underlying structure is that of a Linear Program, but optimization may be used successfully when the non-linearities are separable into functions of just a few variables.

Note that in this book we use the terms Integer Programming (IP) and Mixed Integer Programming interchangeably. Strictly speaking MIP models have both discrete entities and the continuous variables, and pure IP models only have discrete entities. But it seems pedantic to distinguish these two cases unless the distinction is important to our modeling or optimization; and it is not.

The first Section 3.1 of this chapter introduces the different types of IP modeling objects (also referred to as 'global entities'). The following Section 3.2 describes the branch and bound method for solving MIP problems. Binary variables merit special attention: they may be used to formulate do/don't do decisions, implications and other logical expressions, and even products between several binary variables (Section 3.3). As shown in Section 3.4, it would indeed be possible to reduce the whole set of IP modeling objects to binary variables — but this at the expense of efficiency. In combination with real variables, binary variables can be used, among others, to express discontinuities (Section 3.5).

3.1 IP modeling objects: 'global entities'

In principle, all you need in building MIP models are continuous variables and binary variables, where binary variables are decision variables that can take either the value 0 or the value 1. They are often called '0/1 variables' or 'do/don't do' variables. But it is convenient to extend the set of modeling entities to embrace objects that frequently occur in practice.

- **Integer variables**: an integer variable is a variable that must take only an integer (whole number) value.
- **Partial integer variables**: a partial integer variable is a variable that must take an integer value if it is less than a certain user-specified limit L, or any value above that limit.
- **Semi-continuous (SC) variables**: an SC variable is a decision variable that can take either the value 0, or a value between some user specified lower limit L and upper limit U. SCs help model situations where if a variable is to be used at all, it has to be used at some minimum level.
- **Semi-continuous integer variables (SI)**: decision variables that can take either the value 0, or an integer value between some lower limit and upper limit. SIs help model situations where if a variable is to be used at all, it has to be used at some minimum level, and has to be integer.
- **Special Ordered Sets of type 1 (SOS1 or S1)**: an ordered set of non-negative variables at most one of which can take a non-zero value.

- **Special Ordered Sets of type 2 (SOS2 or S2)**: an ordered set of non-negative variables, of which at most two can be non-zero, and if two are non-zero these must be consecutive in their ordering.

Why might some of these variables and sets, which collectively we call global entities, be used? We shall give a brief overview, and later we shall expand on these examples.

Integer variables occur naturally, being most frequently found where the underlying decision variable really has to take on a whole number value for the optimal solution to make sense. If we are modeling the number of trains that leave a railway station in a half-hour interval, then the variable representing this number must take on an integer value — it makes no sense to say that 11.23 trains are to leave.

The idea of **partial integers** arose from a study where car engines in short supply were being allocated to different regions. Some of the regions were large, and took many engines, so it was permissible to round a solution saying the region should take 1654.19 engines to the integer value 1654. But for a regions taking only a few engines, it was not acceptable to round 6.32 engines to 6 engines. It is up to the decision maker to decide the limit L below which it is not acceptable to round a fractional answer to a nearby integer. The modeling system allows the decision maker to specify a different L for each partial integer variable, if so desired. So partial integers provide some computational advantages in problems where it is acceptable to round the LP solution to an integer if the optimal value of a decision variable is quite large, but unacceptable if it is small.

Semi-continuous variables are useful where, if some variable is to be used at all, its value must be no less than some minimum amount. For example, in a blending problem we might be constrained by practical reasons either to use none of a component or at least 20 kg of it. This type of constraint occurs very frequently in practical problems.

Semi-continuous integer variables often arise in shift planning modeling. If we are going to set up a machine, then we have to run it for a whole number of shifts, and in any case for at least some minimum number of shifts. For example, it might be possible only to run the machine for 0, or 12, 13, 14, ... shifts, with the case of running it for 1, 2, ..., 11 shifts making no economic sense.

Special Ordered Sets of type 1 are often used in modeling choice problems, where we have to select at most one thing from a set of items. The choice may be from such sets as: the time period in which to start a job; one of a finite set of possible sizes for building a factory; which machine type to process a part on, *etc.*

Special Ordered Sets of type 2 are typically used to model non-linear functions of a variable. They are the natural extension of the concepts of Separable Programming, but when embedded in a Branch and Bound code (see below Section 3.2) enable truly global optima to be found, and not just local optima. (A local optimum is a point where all the nearest neighbors are worse than it, but where we have no guarantee that there is not a better point some way away. A global optimum is a point which we know to be the best. In the Himalayas the summit of K2 is a local maximum height, whereas the summit of Everest is the global maximum height.)

Theoretically, models that can be built with any of the entities we have listed above can equally well be modeled solely with binary variables. The reason why modern IP systems have some or all of the extra entities is that they often provide significant computational savings in computer time and storage when trying to solve the resulting model. Most books and courses on Integer Programming do not emphasize this point adequately. We have found that careful use of the non-binary global entities often yields very considerable reductions in solution times over ones that just use binary variables.

3.2 IP solving: the ideas behind Branch and Bound

We have seen that a rich modeling environment supplies more than just binary variables, and we shall later see examples of using these extended entities, but we start the discussion of IP modeling by showing the use of binary variables.

3.3.1 Do/don't do decisions

The most common use of binary variables is when we are modeling activities that either have to be done in their entirety or not done at all. For instance, we might want to model whether we drive a particular truck between city *A* and city *B* on a particular day. In this case we either do drive the truck or we do not drive the truck at all: there are just two options. Typically we model the option of doing something by having a binary variable being equal to 1, and the option of not doing the thing with the binary variable being equal to 0. Thus we just have two choices, of which we must select one.

Associated with the activity and with its binary variable, there may be some other properties. Suppose for instance we are considering what we do with the three trucks that we own. Truck 1 has a capacity of 20 tonnes, truck 2 has a capacity of 25 tonnes and truck 3 has a capacity of 30 tonnes. If on a particular day we decide to send truck 1 to a customer then we either send the entire truck or we do not send the entire truck. So we might have a decision variable (a binary variable) $send_1$ taking the value 1 if we send truck 1 out or 0 if we do not send truck 1. Similarly we will have another binary variable $send_2$ which is 1 if truck 2 goes out and 0 otherwise; and variable $send_3$ taking on the value 1 if truck 3 goes out, and 0 otherwise.

Elsewhere in the model we will probably be interested in the total tonnage that goes out on that particular day and we can see that this is

$$20 \cdot send_1 + 25 \cdot send_2 + 30 \cdot send_3$$

because if truck 1 goes out ($send_1 = 1$) it takes 20 tonnes, if truck 2 goes out it independently takes 25 tonnes, and if truck 3 goes out it carries 30 tonnes, so the total is $20 \cdot send_1 + 25 \cdot send_2 + 30 \cdot send_3$. Suppose that we have a decision variable called *out* which gives the total tonnage leaving our depot on that particular day. Then we have the equation

$$out = 20 \cdot send_1 + 25 \cdot send_2 + 30 \cdot send_3$$

3.3.2 Logical conditions

To show many uses of binary variables we will take as an example a set of projects that we may or may not decide to do. We shall call the projects rather unimaginatively A, B, C, D, E, F, G and H and with each of these projects we will associate a decision variable (a binary variable) which is 1 if we decide to do the project and 0 if we decide not to do the project. We call the corresponding variables a, b, c, d, e, f, g and h. So decision variable a taking on the value 1 means that we do project A, whilst a taking on the value of 0 means that we do not do project A. We are now going to express some constraints in words and see how they can be modeled using these binary variables.

3.3.2.1 Choice among several possibilities

The first constraint that we might impose is 'we must choose no more than one project to do'. We can easily see that this can be expressed by the constraint

$$a + b + c + d + e + f + g + h \leq 1$$

Why is this true? Think of selecting one project, for instance project C. If c is 1 then the constraint immediately says that a, b, d, e, f, g and h must be 0 because there is no other setting for the values of these variables that will satisfy that constraint. So if c is 1 then all the others are 0 and, as there is nothing special about C, we can see immediately that the constraint means that we can only do one project.

If the constraint was that we could do no more than three projects then obviously all we have to do in the constraint above is to replace the 1 by 3 and then we can easily see that in fact we can have no more than three of the 0-1 variables being equal to 1. It is possible to have just two of them or one of them or even none of them being equal to 1 but we can certainly not have four or more being 1 and satisfy the constraint at the same time.

Now suppose that the constraint is that we must choose exactly two of the projects. Then the constraint we can use to model this is

$$a + b + c + d + e + f + g + h = 2$$

Since the binary variables can take only the values 0 or 1, we must have exactly two of them being 1 and all the rest of them being 0, and that is the only way that we can satisfy that constraint.

3.3.2.2 Simple implications

Now consider an entirely different sort of constraint. Suppose that 'if we do project A then we must do project B'. How do we model this? Like a lot of mathematics, it is a question of learning a trick that has been discovered. Consider the following constraint

$$b \geq a$$

To show that this formulation is correct, we consider all the possible combinations of settings (there are four of them) of a and b. First, what happens if we do not do project A. If we also do not do project B then the constraint is satisfied ($0 = b \geq a = 0$) and our word constraint is satisfied. If, on the other hand, we do do project B, then $1 = b \geq a = 0$, and the constraint is again satisfied. Now consider what happens if we actually do project A, *i.e.* $a = 1$. Then the constraint is violated if $b = 0$ and is satisfied (0 is not ≥ 1) if $b = 1$, in other words we do project B. The last case to consider is if we do not do project A, *i.e.* $a = 0$ and we do project B, *i.e.* $b = 1$. Then the constraint is satisfied ($1 \geq 0$) and the word constraint is indeed satisfied. We can lay these constraints out in a table. There are only four possible conditions: {do A, do B}, {do A, do not do B}, {do not do A, do not do B}, {do not do A, do B}, and we can see that the illegal one is ruled out by the algebraic constraint.

Table 3.1: Evaluation of a binary implication constraint

$b \geq a$	a=0	a=1
b=0	Yes	No
b=1	Yes	Yes

The next word constraint we consider is 'if we do project A then we must not do project B'. How might we set about modeling this? The way to think of it is to notice that the property of *notdoing* B can be modeled very easily when we already have a binary variable b representing doing B. We invent a new variable

$$\overline{b} = 1 - b$$

where $\overline{b}$ represents the doing of project $\overline{B}$ *i.e.*, the project 'not doing B'. If $b = 1$ then $\overline{b} = 0$ (in other words, if we do project B then we do not do 'not B') whereas if $b = 0$ then $\overline{b} = 1$ (if we do not do project B then we do project $\overline{B}$). This is very convenient and $\overline{b}$ is called the **complement** of b. We can use this trick frequently. Just above we learned how to model 'if we do A then we must do B' and now we are trying to model 'if we do A then we must not do B', *i.e.* 'if we do A then we must do $\overline{B}$'. As 'if we do A then we must do B' was modeled by $b \geq a$ we can immediately see that the constraint 'if we do A then we must do $\overline{B}$' can be obtained by replacing b by $\overline{b}$ in the constraint, in other words $\overline{b} \geq a$. Replacing $\overline{b}$ by $1 - b$ we get

$$1 - b \geq a$$

or

$$1 \geq a + b$$

or alternatively

$$a + b \leq 1$$

Now that we have obtained this constraint, it is quite obvious. What it says is that if we do project A ($a = 1$) then b must be 0. This is exactly what we wanted to model. The point of the somewhat long-winded argument we showed above, however, is that we have used the result from the first logical constraint that we wanted to model, plus the fact that we have now introduced the notion of the complement of the project, to build up a newer and more complicated constraint from these two primitive concepts. This is what we will do frequently in what follows.

We see an example of this by trying to model the word constraint 'if we do not do A then we must do B', in other word, 'if not A then B'. We can go back immediately to our first logical constraint 'if we do A then we must do B' which was modeled as $b \geq a$. Now we are actually replacing A by not A, so we can see immediately that our constraint is

$$b \geq 1 - a$$

which is

$$a + b \geq 1$$

Again this constraint is obvious now that we have got to it. If we do not do A then $a = 0$, so $b \geq 1$ and since the maximum value of b is 1 then this immediately means that $b = 1$. Again we have just taken our knowledge of how to model 'if A then B' and the notion of the complement of a variable to be able to build up a more complex constraint.

The next constraint to consider is 'if we do project A we must do project B, and if we do project B we must do project A'. We have seen that the first is modeled as $b \geq a$ and the second as $a \geq b$. Combining these two constraints we get

$$a = b$$

in other words projects A and B are selected or rejected together, which is exactly what we expressed in our word constraint.

3.3.2.3 Implications with three variables

The next constraint we consider is 'if we do project A then we must do project B and project C'. The first thing to note is that this is in fact two constraints. One, the first, is 'if we do A then we must do project B' and the second constraint is 'if we do A then we must do project C'. With this observation we can see that the word constraint can be modeled by the two inequalities

$$b \geq a \text{ and } c \geq a$$

so that if $a = 1$, then both $b = 1$ and $c = 1$.

Another constraint might be 'if we do project A then we must do project B or project C'. It is like the previous constraint, except we now have an 'or' in place of the 'and'. The constraint

$$b + c \geq a$$

models this correctly. To see this, consider the following situation. If a is 0 then $b + c$ can be anything, and so b and c are not constrained. If $a = 1$, then one or both of b and c must be 1.

We may also try to model the inverse situation: 'if we do Project B or project C then we must do A'. This is again a case that may be formulated as two separate constraints: 'if we do B then we must do A' and 'if we do C then we must do A', giving rise to the following two inequalities

$$a \geq b \text{ and } a \geq c$$

so that if either $b = 1$ or $c = 1$, then we necessarily have $a = 1$.

A harder constraint to model is the following 'if we do both B and C then we must do A'. How might we model this? One way to think about it is to express it in the following way: 'if we do both B and C then we must not do not-A', or, 'we can do at most two of B, C or not-A' which we would model as

$$b + c + (1 - a) \leq 2$$

or in other words

$$b + c - a \leq 1$$

or perhaps more conventionally

$$a \geq b + c - 1$$

Looking at this last inequality, we can see that there is no effect on a when b and c are 0, or when just one of b and c is 1, but a does have to be ≥ 1 when both b and c are 1. A binary variable having to be greater than or equal to 1 means that the binary variable has to be precisely 1.

3.3.2.4 Generalized implications

Generalizing the previous we now might try to model 'if we do two or more of B, C, D or E then we must do A', and our first guess to this might be the constraint

$$a \geq b + c + d + e - 1$$

Certainly if, say, b and c are both equal to 1 then we have $a \geq 1$ and so a must equal 1, but the problem comes when three of the variables are equal to 1, say b, c and d. Then the constraint says that $a \geq 3 - 1$, *i.e.* $a \geq 2$, which is impossible as a is a binary variable. So we have to modify the constraint as follows

$$a \geq \frac{1}{3} \cdot (b + c + d + e - 1)$$

The biggest value that the expression inside the parentheses can take is 3, if $b = c = d = e = 1$. The $\frac{1}{3}$ in front of the parenthesis means that in the worst case a must be ≥ 1 (so a is equal to 1). But we must verify that the constraint is true if, say, just $b = c = 1$ (and d and e are equal to 0). In this case we have that $a \geq \frac{1}{3} \cdot (1 + 1 + 0 + 0 - 1)$ *i.e.* $a \geq \frac{1}{3}$. But since a can only take on the values 0 or 1 then the constraint $a \geq \frac{1}{3}$ means that a must be 1. This is exactly what we want.

We can generalize this to modeling the statement 'if we do M or more of N projects (B, C, D, ...) then we must do project A' by the constraint

$$a \geq \frac{b + c + d + \ldots - M + 1}{N - M + 1}$$

So far we have only given one way of modeling each of these constraints. We return to the constraint 'if we do B or C then we must do A' which we modeled as two constraints $a \geq b$ and $a \geq c$. It is possible to model this with just one constraint if we take this as a special case of the 'M or more from N' constraint we have just modeled. 'If we do B or C then we must do A' is the same as 'if we do M or more of N projects (B, C, D, ...) then we must do project A' with $M = 1$ and $N = 2$. So the constraint is

$$a \geq \frac{b + c - M + 1}{N - M + 1} \quad \text{i.e.}$$

$$a \geq \frac{b + c - 1 + 1}{2 - 1 + 1} \quad \text{i.e.}$$

$$a \geq \frac{1}{2} \cdot (b + c)$$

So this single constraint is exactly the same in terms of binary variables as the two constraints which we produced before. Which of these two representations is better? In fact the representation in terms of two constraints is better. But both of the two are correct and both will give a correct answer if put into an Integer Programming system. It is just that the first pair of constraints will in general give a solution more rapidly.

More and more complicated constraints can be built up from the primitive ideas we have explored so far. Since these more complicated constraints do not occur very frequently in actual practical modeling we shall not explore them further. Table 3.2 summarizes the formulations of logical conditions we have seen in the different paragraphs of Section 3.3.2.

Table 3.2: Formulation of logical conditions using binary variables

At most one of A, B,...,H	$a+b+c+d+e+f+g+h \leq 1$
Exactly two of A, B,...,H	$a+b+c+d+e+f+g+h = 2$
If A then B	$b \geq a$
Not B	$\overline{b} = 1-b$
If A then not B	$a+b \leq 1$
If not A then B	$a+b \geq 1$
If A then B, and if B then A	$a = b$
If A then B and C	$b \geq a$ and $c \geq a$
If A then B or C	$b+c \geq a$
If B or C then A	$a \geq b$ and $a \geq c$
	or alternatively: $a \geq \frac{1}{2} \cdot (b+c)$
If B and C then A	$a \geq b+c-1$
If two or more of B, C, D or E then A	$a \geq \frac{1}{3} \cdot (b+c+d+e-1)$
If M or more of N projects (B, C, D, ...) then A	$a \geq \frac{b+c+d+\ldots-M+1}{N-M+1}$

3.3.3 Products of binary variables

We move on to modeling the product of binary variables. Suppose we have three binary variables b_1, b_2 and b_3 and we want to model the equation

$$b_3 = b_1 \cdot b_2$$

This is not a linear equation since it involves the product of two variables, so we have to express it in some linear form. There is a trick and it is another one of these tricks that one just has to learn. Consider the following set of three inequalities

$$b_3 \leq b_1$$
$$b_3 \leq b_2$$
$$b_3 \geq b_1 + b_2 - 1$$

Then we claim that this represents the product expression that we wish to model. To see we construct the following Table 3.3.

Table 3.3: Product of two binaries

b_1	b_2	b_3	$b_3 = b_1 \cdot b_2$?	$b_3 \leq b_1$?	$b_3 \leq b_2$?	$b_3 \geq b_1 + b_2 - 1$?
0	0	0	Yes	Yes	Yes	Yes
0	0	1	No	No	No	Yes
0	1	0	Yes	Yes	Yes	Yes
0	1	1	No	No	Yes	Yes
1	0	0	Yes	Yes	Yes	Yes
1	0	1	No	Yes	No	Yes
1	1	0	No	Yes	Yes	No
1	1	1	Yes	Yes	Yes	Yes

We can see that the column headed $b_3 = b_1 \cdot b_2$? is true if and only if we have a 'Yes' in the three columns $b_3 \leq b_1$?, $b_3 \leq b_2$? and $b_3 \geq b_1 + b_2 - 1$?, so the three linear equations do exactly represent and are true at exactly the same time as the product is true.

This is a particularly long winded way of demonstrating the equivalence of the product term and the three linear equations and in fact now we have got it it is actually quite easy to see why these three inequalities are correct. Since b_3 is b_1 multiplied by something that is less than or equal to 1, b_3 will always be less than or equal to b_1 and by a similar argument b_3 will always be less than or equal to b_2. The only further case we have to consider is when both b_1 and b_2 are equal to 1 and then we have to force b_3 to be equal to 1. This is done by the constraint $b_3 \geq b_1 + b_2 - 1$ which is non restrictive if only one or none of b_1 and b_2 are 1 but forces b_3 to be 1 when $b_1 = b_2 = 1$.

Looking at the constraint this way immediately enables us to model for instance

$$b_4 = b_1 \cdot b_2 \cdot b_3,$$

in other words the product of three variables, as the four constraints

$$b_4 \leq b_1$$

$$b_4 \leq b_2$$
$$b_4 \leq b_3$$
$$b_4 \geq b_1 + b_2 + b_3 - 2$$

If any of b_1, b_2 and b_3 are 0 then b_4 must be 0 but if b_1, b_2 and b_3 are 1 then b_4 must be greater than or equal to $3 - 2$, *i.e.* b_4 must be greater than or equal to 1 so b_4 must be 1.

3.3.4 Dichotomies: either/or constraints

All the constraints we have seen so far have had to be satisfied simultaneously, but sometimes we need to model that either Constraint 1 or Constraint 2 has to be satisfied, not necessarily both of them. Consider the problem:

minimize	$Z = x_1 + x_2$
subject to	$x_1 \geq 0, x_2 \geq 0$ and
	Either $2 \cdot x_1 + x_2 \geq 6$ (Constraint 1) or $x_1 + 2 \cdot x_2 \geq 7$ (Constraint 2)

Since we have just two variables, we can graph the feasible region, and we have done this in the figure below where the grey shaded area is the feasible region.

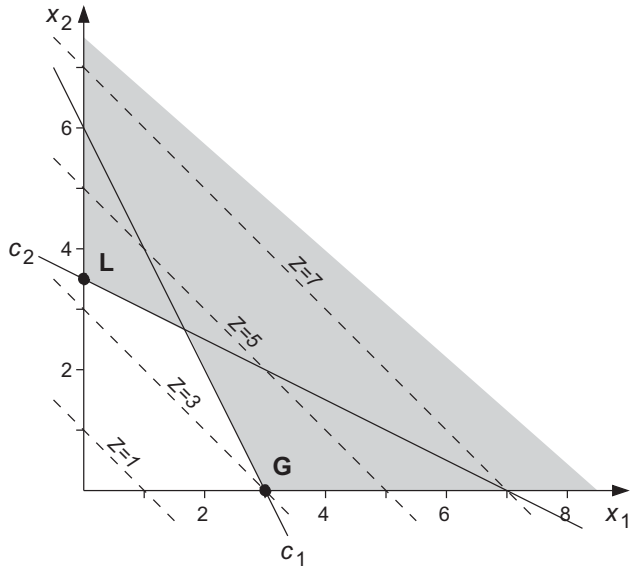

Figure 3.2: Example of either/or constraints

Point L ($x_1 = 0, x_2 = 3.5, Z = 3.5$) is a local optimum. Point G ($x_1 = 3, x_2 = 3, Z = 3$) is the global minimum.

We can model this with one additional binary variable b.

$$2 \cdot x_1 + x_2 \geq 6 \cdot b$$
$$x_1 + 2 \cdot x_2 \geq 7 \cdot (1 - b)$$

To show that this is true, we just have to consider the two cases:

$b = 0$:	$2 \cdot x_1 + x_2 + x_3 \geq 0$	$x_1 + 2 \cdot x_2 + 3 \cdot x_3 \geq 7$	Constraint 2 is satisfied
$b = 1$:	$2 \cdot x_1 + x_2 + x_3 \geq 6$	$x_1 + 2 \cdot x_2 + 3 \cdot x_3 \geq 0$	Constraint 1 is satisfied

3.4 Binary variables 'do everything'

All global entities (general integers, partial integers, semi-continuous variables, and both sorts of Special Ordered Sets) can be expressed in terms of binary variables. However, as shown by the examples in this section, it is usually preferable to use the specific global entities.