Integer programming solution methods - introduction **J E Beasley**

Capital budgeting

There are four possible projects, which each run for 3 years and have the following characteristics.

			Capital requirements (£m)		
Project	Return (£m)	Year	1	2	3
1	0.2		0.5	0.3	0.2
2	0.3		1.0	0.8	0.2
3	0.5		1.5	1.5	0.3
4	0.1		0.1	0.4	0.1
Available capital (£m)			3.1	2.5	0.4

We have a **decision problem** here: **Which projects would you choose in order to maximise the total return?**

Capital budgeting solution

We follow the formulation approach below:

- variables
- constraints
- objective.

Variables

Here we are trying to decide whether to undertake a project or not (a "go/no-go" decision). One "trick" in formulating IP's is to introduce variables which take the integer values 0 *or* 1 and represent *binary* decisions (e.g. do a project or not do a project) with typically:

- the positive decision (do something) being represented by the value 1; and
- the negative decision (do nothing) being represented by the value 0.

Such variables are often called *zero-one* or *binary* variables

To define the variables we use the *verbal* description of

x_j = 1 if we decide to do project j (j=1,...,4)
= 0 otherwise, i.e. not do project j (j=1,...,4)

Note here that, by definition, the x_j are *integer* variables which must take one of two possible values (zero or one).

Constraints

The constraints relating to the availability of capital funds each year are

$$0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 \leq 3.1 \quad \text{(year 1)}$$
$$0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 \leq 2.5 \quad \text{(year 2)}$$
$$0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 \leq 0.4 \quad \text{(year 3)}$$

Objective

To maximise the total return - hence we have

maximise $0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$

This gives us the complete IP which we write as

maximise $0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$

subject to

$$0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 \leq 3.1$$
$$0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 \leq 2.5$$
$$0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 \leq 0.4$$
$$x_j = 0 \text{ or } 1 \qquad j=1,...,4$$

Note:

- in writing down the complete IP we include the information that $x_j = 0$ or 1 (j=1,...,4) as a reminder that the variables are integers
- you see the usefulness of defining the variables to take zero/one values - e.g. in the objective the term $0.2x_1$ is zero if $x_1=0$ (as we want since no return from project 1 if we do not do it) and 0.2 if $x_1=1$ (again as we want since get a return of 0.2 if we do project 1). Hence effectively the zero-one nature of the decision variable means that we always capture in the single term $0.2x_1$ what happens both when we do the project and when we do not do the project.
- you will note that the objective and constraints are linear (i.e. any term in the constraints/objective is either a constant or a constant multiplied by an unknown). In this course we deal only with linear integer programs (IP's with a linear objective and linear constraints). It is plain though that there do exist non-linear integer programs - these are, however, outside the scope of this course.

Extensions to this basic problem include:

- projects of different lengths
- projects with different start/end dates
- adding capital inflows from completed projects
- projects with staged returns

- carrying unused capital forward from year to year
- mutually exclusive projects (can have one or the other but not both)
- projects with a time window for the start time.

In fact note here that integer programming/quantitative modelling techniques are increasingly being used for financial problems.

Solving IP's

Solution methods for IP's can be categorised as:

- *optimal*
- *heuristic*

An optimal algorithm is one which (mathematically) *guarantees* to find the optimal solution.

It may be that we are not interested in the optimal solution:

- because the size of problem that we want to solve is beyond the computational limit of known optimal algorithms within the computer time we have available; or
- we could solve optimally but feel that this is not worth the effort (time, money, etc) we would expend in finding the optimal solution.

In such cases we can use a heuristic algorithm - that is an algorithm that should hopefully find a feasible solution which, in objective function terms, is close to the optimal solution. In fact it is often the case that a well-designed heuristic algorithm can give good quality (near-optimal) results.

Note here that the methods presented below are suitable for solving both IP's (all variables integer) and MIP's (mixed-integer programs - some variables integer, some variables allowed to take fractional values).

General purpose optimal solution algorithms

We shall deal with just two general purpose (able to deal with any IP) optimal solution algorithms for IP's:

- enumeration (sometimes called complete enumeration)
- branch and bound (tree search).

We consider each of these in turn below. Note here that there exists another general purpose solution algorithm based upon *cutting planes.*

Enumeration

Unlike LP (where variables take continuous values (≥ 0)) in IP's (where all variables are integers) each variable can only take a finite number of discrete (integer) values.

Hence the obvious solution approach is simply to *enumerate* all these possibilities - calculating the value of the objective function at each one and choosing the (*feasible*) one with the optimal value.

For example for the capital budgeting problem considered above there are $2^4=16$ possible solutions. These are:

x_1	x_2	x_3	x_4	
0	0	0	0	do no projects
0	0	0	1	do one project
0	0	1	0	
0	1	0	0	
1	0	0	0	
0	0	1	1	do two projects
0	1	0	1	
1	0	0	1	
0	1	1	0	
1	0	1	0	
1	1	0	0	
1	1	1	0	do three projects
1	1	0	1	
1	0	1	1	
0	1	1	1	
1	1	1	1	do four projects

Hence for our example we merely have to examine 16 possibilities before we know precisely what the best possible (optimal) solution is. This example illustrates a general truth about integer programming:

What makes solving the problem easy when it is small is precisely what makes it become very hard very quickly as the problem size increases

This is simply illustrated: suppose we have 100 integer variables each with 2 possible integer values then there are $2\times2\times2\times \ldots \times2 = 2^{100}$ (approximately 10^{30}) possibilities which we have to enumerate (obviously many of these possibilities will be infeasible, but until we generate one we cannot check it against the constraints to see if it is feasible or not).

This number is plainly too many for this approach to solving IP's to be computationally practicable. To see this consider the fact that the universe is around 10^{10} years old so we

would need to have considered 10^{20} possibilities per year, approximately 4×10^{12} possibilities per second, to have solved such a problem by now if we started at the beginning of the universe.

Be clear here - **conceptually** there is not a problem - simply enumerate all possibilities and choose the best one. But **computationally** (numerically) this is just impossible.

IP nowadays is often called "*combinatorial optimisation*" indicating that we are dealing with optimisation problems with an extremely large (combinatorial) increase in the number of possible solutions as the problem size increases.

Branch and bound (tree search)

The most effective general purpose optimal algorithm is LP-based tree search (tree search also being called branch and bound). *This is a way of systematically enumerating feasible solutions such that the optimal integer solution is found.*

Where this method differs from the enumeration method is that *not all* the feasible solutions are enumerated but only a fraction (hopefully a small fraction) of them. However we can still *guarantee* that we will find the optimal integer solution. The method was first put forward in the early 1960's by Land and Doig.

Consider our example capital budgeting problem. What made this problem difficult was the fact that the variables were restricted to be integers (zero or one). If the variables had been allowed to be fractional (e.g. allowed to take all values between zero and one) then we would have had an LP which we could easily solve.

Relaxing the integrality requirement to its continuous equivalent gives us the LP relaxation of the problem [here replace $x_j = 0$ or 1 j=1,...,4 by $0 \le x_j \le 1$ j=1,...,4].

This LP relaxation is

maximise $0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$

subject to

$$0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 \le 3.1$$
$$0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 \le 2.5$$
$$0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 \le 0.4$$
$$0 \le x_j \le 1 \qquad j=1,...,4$$

Suppose that we were to solve this LP. Then we get $x_2=0.5$, $x_3=1$, $x_1=x_4=0$ of value 0.65 (i.e. the objective function value of the optimal linear programming solution is 0.65).

As a result of this we now know something about the optimal integer solution, namely that it is ≤ 0.65, i.e. this value of 0.65 is a (upper) *bound* on the optimal integer solution. This is because when we relax the integrality constraint we (as we are maximising) end up with a solution value at least that of the optimal integer solution (and maybe better).

Consider this LP relaxation solution. We have a variable x_2 which is fractional when we need it to be integer.

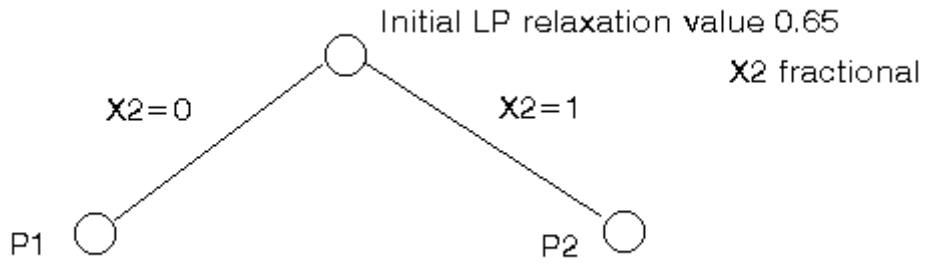
How can we rid ourselves of this troublesome fractional value?

To remove this troublesome fractional value we can generate two new problems:

- original LP relaxation plus $x_2=0$
- original LP relaxation plus $x_2=1$

then we will claim that the optimal integer solution to the original problem is contained in one of these two new problems. **Why is this true?**

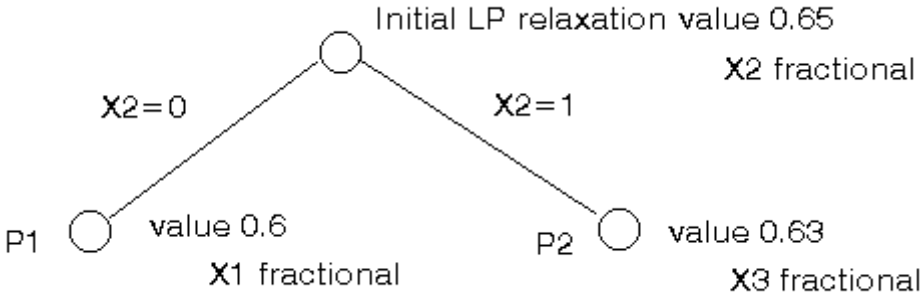
This process of taking a fractional variable (a variable which takes a fractional value in the LP relaxation) and explicitly constraining it, to in this case each of its two integer values, is known as ***branching***. It can be represented diagrammatically as below (in a tree diagram, which is how the name ***tree search*** arises).



We now have two new LP relaxations to solve. If we do this we get:

- P1 - original LP relaxation plus $x_2=0$, solution $x_1=0.5$, $x_3=1$, $x_2=x_4=0$ of value 0.6
- P2 - original LP relaxation plus $x_2=1$, solution $x_2=1$, $x_3=0.67$, $x_1=x_4=0$ of value 0.63

This can be represented diagrammatically as below.

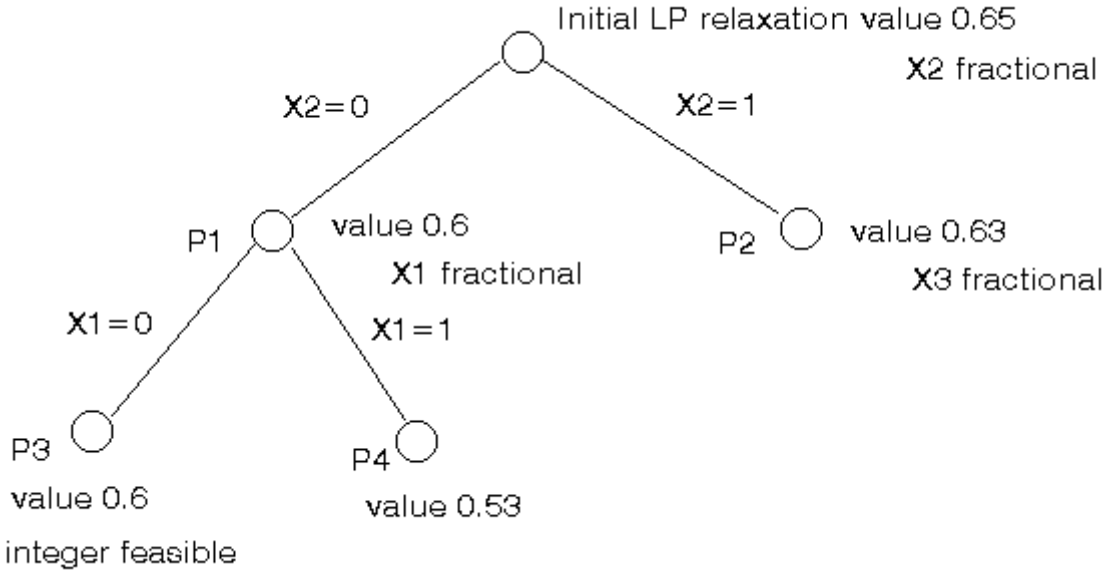


To find the optimal integer solution we just repeat the process, choosing one of these two problems, choosing one fractional variable and generating two new problems to solve.

Choosing problem P1 we branch on x_1 to get our list of LP relaxations as:

- P3 - original LP relaxation plus $x_2=0$ (P1) plus $x_1=0$, solution $x_3=x_4=1$, $x_1=x_2=0$ of value 0.6
- P4 - original LP relaxation plus $x_2=0$ (P1) plus $x_1=1$, solution $x_1=1$, $x_3=0.67$, $x_2=x_4=0$ of value 0.53
- P2 - original LP relaxation plus $x_2=1$, solution $x_2=1$, $x_3=0.67$, $x_1=x_4=0$ of value 0.63

This can again be represented diagrammatically as below.



At this stage we have identified a integer feasible solution of value 0.6 at P3. There are no fractional variables so no branching is necessary and P3 can be dropped from our list of LP relaxations.

Hence we now have new information about our optimal (best) integer solution, namely that it lies between 0.6 and 0.63 (inclusive). **Why is this true?**

Consider P4, it has value 0.53 and has a fractional variable (x_3). However if we were to branch on x_3 any objective function solution values we get after branching can never be better (higher) than 0.53. As we already have an integer feasible solution of value 0.6 P4 can be dropped from our list of LP relaxations since branching from it could never find an improved feasible solution. This is known as ***bounding*** - using a known feasible solution to identify that some relaxations are not of any interest and can be discarded.

Hence we are just left with:

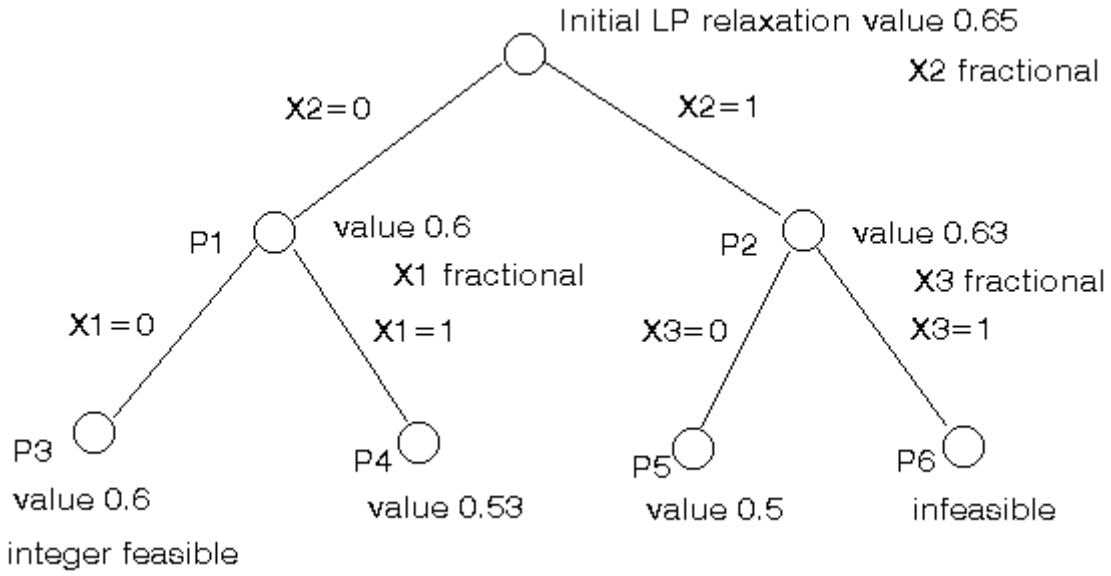
- P2 - original LP relaxation plus $x_2=1$, solution $x_2=1$, $x_3=0.67$, $x_1=x_4=0$ of value 0.63

Branching on x_3 we get

- P5 - original LP relaxation plus $x_2=1$ (P2) plus $x_3=0$, solution $x_1=x_2=1$, $x_3=x_4=0$ of value 0.5
- P6 - original LP relaxation plus $x_2=1$ (P2) plus $x_3=1$, problem infeasible

Neither of P5 or P6 lead to further branching so we are done, we have discovered the optimal integer solution of value 0.6 corresponding to $x_3=x_4=1$, $x_1=x_2=0$.

The entire process we have gone through to discover this optimal solution (and to prove that it is optimal) is shown graphically below.



You should be clear as to why 0.6 is the optimal integer solution for this problem, simply put if there were a better integer solution the above tree search process would (logically) have found it.

Note here that this method, like complete enumeration, also involves powers of two as we progress down the (binary) tree. However also note that we did not enumerate all possible integer solutions (of which there are 16). Instead here we solved 7 LP's. This is an important point, and indeed why tree search works at all. **We do not need to examine as many LP's as there are possible solutions.** Whilst the computational efficiency of tree search differs for different problems it is this basic fact that enables us to solve problems that would be completely beyond us were we to try complete enumeration.

You may have noticed that in the example above we never had more than one fractional variable in the LP solution at any tree node. This arises due to the fact that in constructing the above example I decided to make the situation as simple as possible. In general we might well have more than one fractional variable at a tree node and so we face a decision as to which variable to choose to branch on. A simple rule for deciding might be to take the fractional variable which is closest in value to 0.5, on the basis that the two branches (setting this variable to zero and one respectively) may well perturb the situation significantly.

Good computer packages (solvers) exist for finding optimal solutions to IP's/MIP's via LP-based tree search. Many of the computational advances in IP optimal solution methods (e.g. constraint aggregation, coefficient reduction, problem reduction, automatic generation of valid inequalities) are included in these packages. Often the key to making successful use of such packages for any particular problem is to put effort into a good formulation of the problem in terms of the variables and constraints. By this we mean that for any particular IP there may be a number of valid formulations. Deciding which formulation to adopt in a solution algorithm is often a combination of experience and trial and error.

Constraint logic programming (CLP), also called constraint programming, which is essentially branch and bound but without the bound, can be of use here if:

- the problem cannot be easily expressed in linear mathematics
- the gap between the LP relaxation solution and the IP optimal solution is so large as to render LP-based tree search impracticable.

Other approaches

In order to link to the talks given by other speakers I have been asked specifically to mention here:

- dynamic programming – this is a technique, applicable if problems have a certain structure, that enables optimal solutions to be computed. It is very different in nature from branch and bound. Two problems you may have met that use dynamic programming are:
    - shortest path problem (finding the shortest path between two points in a graph)
    - longest path problem (finding the critical path in a PERT/CPM network
- cutting planes. Cutting planes (essentially) start with the LP relaxation, as does branch and bound. However once that LP has been solved then, instead of branching, cutting plane methods attempt to add one or more constraints (cuts) to the LP. These constraints should be such as to:

  (a) perturb the solution (more technically be such as to make the current LP solution infeasible)

  (b) not eliminate the optimal integer solution (i.e. whatever constraint has been added it must be satisfied by the optimal integer solution)

  Once one or more constraints have been added then the LP is resolved and the process repeated.
- Gomory f-cuts, also called Gomory cuts, are cuts (constraints) that can be deduced from the optimal LP solution which will (eventually) lead you to the optimal integer solution. The difficulty with using them in practice is essentially numeric – computers only do arithmetic to a finite number of decimal places and so errors can be accidentally introduced which lead to the method going astray.