Integer Linear Programming

This is the name given to L.P. problems which have the extra constraint that some or all variables have to be integer.

Examples

1. Capital Budgeting

A firm has n projects that it would like to undertake but because of budget limitations not all can be selected. In particular project j is expected to produce a revenue of c_j but requires an investment of a_{ij} in time period i for $i=1,\ldots m$. The capital available in time period i is b_i. The problem of minimising revenue subject to the budget constraints can be formulated as follows: let $x_j = 0$ or 1 correspond to not proceeding or respectively proceeding with project j then we have to

$$\text{Maximise} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i=1,\ldots m$$

$$0 \leqslant x_j \leqslant 1 \qquad x_j \text{ integer} \qquad j=1 \ldots n$$

2. Depot location

We consider here a simple problem of this type: a company has selected m possible sites for distribution of its products in a certain area. There are n customers in the area and the transport cost of supplying the whole of customer j's requirements over the given planning period from potential site i is c_{ij}. Should site i be developed it will cost f_i to construct a depot there. Which sites should be selected to minimise the total construction plus transport cost?

To do this we introduce m variables $y_1, \ldots y_m$ which can only take values 0 or 1 and correspond to a particular site being not developed or developed respectively. We next define x_{ij} to be the fraction of customer j's requirements supplied from depot i in a given solution. The problem can then be expressed.

$$\text{Minimise} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i$$

$$\text{subject to} \quad \sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, \ldots n \qquad *$$

$$x_{ij} \leq y_i \qquad i=1,\ldots m, \; j = 1, \ldots n$$

$$x_{ij} \geq 0 \quad 0 \leq y_i \leq 1 \quad y_i \text{ integer} \qquad \begin{matrix} i=1,\ldots m \\ j=1,\ldots n \end{matrix}$$

Note that if $y_i = 0$ then $f_i y_i = 0$ and there is no contribution to the total cost. Also $x_{ij} \leq y_i$ implies $x_{ij} = 0$ for $j = 1, \ldots n$ and so no goods are distributed from site i. This corresponds exactly to no depot at site i.

On the other hand if $y_i = 1$ then $f_i y_i = f_i$ which is the cost of constructing depot i. Also $x_{ij} \leq y_i$ becomes $x_{ij} \leq 1$ which holds any way from the constraints $*$.

3. Set Covering

Let $S_1, \ldots S_n$ be a family of subsets of a set $S = \{1, 2, \ldots m\}$. A covering of S is a subfamily S_j for $j \in I$ such that $S = \bigcup_{j \in I} S_j$. Assume that each subset S_j has cost $c_j > 0$ associated with it. We define the cost of a cover to be the sum of the costs of the subsets included in the cover.

The problem of finding a cover of minimum cost is of particular practical significance. As an integer program it can be specifed as follows: define the m x n matrix $A = || \; a_{ij} \; ||$ by

$$a_{ij} = 1 \text{ if } i \in S_j$$
$$= 0 \text{ otherwise}$$

Let x_j be 0 - 1 variables with $x_j = 1(0)$ to mean set S_j is included (respectively not included) in the cover. The problem is to

$$\text{minimise} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1,\ldots m \tag{15.1}$$

$$x_j = 0 \text{ or } 1$$

The m inequality constraints have the following significance: since $x_j = 0$ or 1 and the coefficients a_{ij} are also 0 or 1 we see that $\sum_{j=1}^{n} a_{ij} x_j$ can be zero only if $x_j = 0$ for all j such that $a_{ij} = 1$. In other words only if no set S_j is chosen such that $i \in S_j$. The inequalities are put in to aviod this.

As an example consider the following simplified airline crew scheduling problem. An airline has m scheduled flight-legs per week in its current service. A flight-leg being a single flight flown by a single crew e.g. London - Paris leaving Heathrow at 10.30 am. Let S_j $j = 1,\ldots n$ be the collection of all possible weekly sets of flight-legs that can be flown by a single crew. Such a subset must take account of restrictions like a crew arriving in Paris at 11.30 am. cannot take a flight out of New York at 12.00 pm. and so if c_j is the cost of set S_j of flight-legs then the problem of minimising cost subject to covering all flight-legs is a set covering problem. Note that if crews are not allowed to be passengers on a flight, i.e. so that they can be flown to their next flight, then we have to make 15.1 an equality - the set partitioning problem.

General Terminology:

The most general problem called the mixed integer programming problem can be specified as

$$\text{minimise} \quad x_0 = \underline{c} \cdot \underline{x}$$

subject to

$$A\underline{x} = \underline{b}$$
$$x_j \geq 0 \qquad j = 1 \cdots n$$
$$x_j \text{ integer for } j \in IN$$

where IN is some subset of $N_0 = \{0, 1, \cdots n\}$.

When $IN = N_0$ we have what is called a pure integer programming problem. For such a problem one generally has all given quantities c_j, a_{ij}, b_i integer. One has to be careful here. Consider for example

$$\text{minimise} \quad x_0 = -\tfrac{1}{3}x_1 - \tfrac{1}{2}x_2$$

subject to

$$\tfrac{2}{3}x_1 + \tfrac{1}{3}x_2 \leq \tfrac{1}{3}$$
$$\tfrac{1}{2}x_1 - \tfrac{3}{2}x_2 \leq \tfrac{2}{3}$$
$$x_1, x_2 \geq 0 \text{ and integer}$$

As defined this is not a pure problem. For a start x_0 will not necessarily be integer and neither will the slack variables. If we want to use an algorithm for solving pure problems we must scale the objective and constraints to give

minimise $x_0 = -2x_1 - 3x_2$
subject to

$$\begin{aligned} 2x_1 + x_2 + x_3 \quad &= 4 \\ 3x_1 - 9x_2 \quad + x_4 &= 4 \\ x_1, \dots x_4 \geq 0 \text{ and integer.} \end{aligned}$$

A final class of problems is the <u>pure 0-1 programming problem</u>

maximise $x_0 = \underline{c} \cdot \underline{x}$
subject to

$$A\underline{x} \leq \underline{b}$$
$$x_j = 0 \text{ or } 1 \qquad \text{for } j = 1, \dots n.$$

<u>Further Uses of Integer Variables</u>

(1) If a variable x can only take a finite number of values $p_1, \dots p_m$ we can replace x by the expression

$$p_1 w_1 + \dots + p_m w_m$$

where $$w_1 + \dots + w_m = 1$$

and $$w_i = 0 \text{ or } 1 \qquad i = 1, \dots m$$

For example x might be the output of a plant which can be small p_1, medium p_2 or large p_3. The cost c(x) of the plant could be represented by

$$c_1 w_1 + c_2 w_2 + c_3 w_3$$

where c_1 is the cost of a small plant etc.

(2) In L.P. one generally consider all constraints to be holding simultaneously. It is possible that the variable might have to satisfy one or other of a set of constraints

e.g.

(a) $0 \le x \le M$

$0 \le x \le 1$ or $x \ge 2$

can be expressed

$$x \le 1 + (M - 1)\delta$$

$$x \ge 2 + M(\delta - 1)$$

$$x \ge 0 \qquad \delta = 0 \text{ or } 1$$

$x \le M$ is a notional upper bound to make this approach possible.

(b) $x_1 + x_2 \le 4$

$x_1 \ge 1$ or $x_2 \ge 1$ but not both ≥ 1

$x_1, x_2 \ge 0$

can be expressed

$$x_1 + x_2 \le 4$$

$$x_1 \ge \delta$$

$$x_2 \ge 1 - \delta$$

$$x_1 \le (1-\delta) + 4\delta$$

$$x_2 \le \delta + 4(1-\delta)$$

$$\delta = 0 \text{ or } 1$$

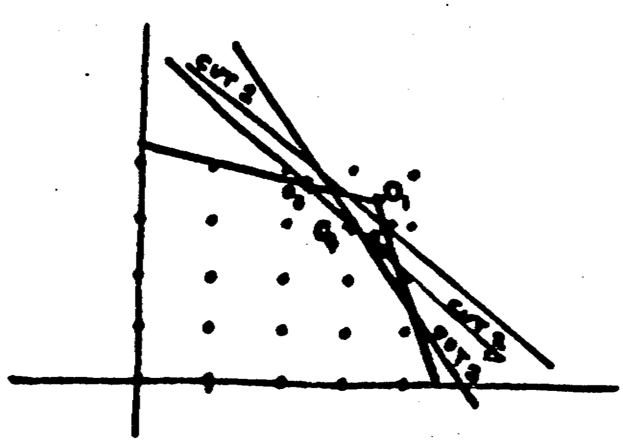
Integer programming problems generally take much longer to solve then the corresponding linear program obtained by ignoring integrality.. It is wise therefore to consider the possibility of solving as a straight forward L.P. and then rounding e.g. in the trim-loss problem. This is not always possible for example if x_1 is a 0 - 1 variable such that $x_1 = 0$ means do not build a plant and $x_1 = 1$ means build a plant then rounding $x_1 = \frac{1}{2}$ is not very satisfactory.

A cutting plane algorithm for the pure problem

The rationale behind this approach is:-

1) Solve the continuous problem as an L.P. i.e. ignore integrality.
2) If by chance the optimal basic variables are all integer then the optimum solution has been found. Otherwise:-
3) Generate a cut i.e. a constraint which is satisfied by all integer solutions to the problem but not by the current L.P. solution.
4) Add this new constraint and go to (1).

The idea of such an approach is illustrated below:-



It is straight forward to show that if at any stage the current L.P. solution x is integer it is the optimal integer solution. This is because x is optimal over a region containing all feasible integer solutions.

The problem is to define cuts that ensure the convergence of the algorithm in a finite number of steps. The first finite algorithm was devised by R.E. Gomory.

It is based on the following construction: let

(15.2) $$a_1 x_1 + \ldots + a_n x_n = b$$

be an equation which is to be satisfied by non-negative integers $x_1, \ldots x_n$ and let S be the set of possible solutions.

For a real number ξ we define $\lfloor \xi \rfloor$ it to be the largest integer $\le \xi$. Thus $\xi = \lfloor \xi \rfloor + \varepsilon$ where $o \le \varepsilon < 1$.

$$\lfloor 6\tfrac{1}{2} \rfloor = 6 \qquad \lfloor 3 \rfloor = 3 \qquad \lfloor -4\tfrac{1}{2} \rfloor = -5$$

Now let $a_j = a_j + f_j$ and $b = b + f$ in (15.2) then we have

$$\sum_{j=1}^{n} (\lfloor a_j \rfloor + f_j) x_j = \lfloor b \rfloor + f$$

and hence

(15.3) $$\sum_{j=1}^{n} f_j x_j - f = b - \sum_{j=1}^{n} a_j x_j$$

Now for $x \in S$ the right hand side of 15.3 is clearly integer and so $\xi = \sum f_j x_j - f$ is integer for $x \in S$. Since $x \ge 0$ for $x \in S$ we also have $\xi \ge - f > - 1$ and since ξ is integer we deduce that $\xi \ge 0$ and that

$$\sum_{j=1}^{n} f_j x_j \ge f \qquad \text{for } x \in S$$

Suppose now that one has solved the continuous problem in (1) of our cutting plane algorithm and the solution is not integer. Therefore there is a basic variable x_i with

$$x_i + \sum_{j \notin I} b_{ij} x_j = b_{io}$$

where b_{io} is not integer.

Putting $f_i = b_{ij} - \lfloor b_{ij} \rfloor$ and $f = b_{io} - \lfloor b_{io} \rfloor$ and we deduce that

(15.4) $$\sum_{j \notin I} f_j x_j \geq f$$

for all integer solutions to our problem.

Now $f > 0$ since b_{io} is not integer and so (15.4) is not satisfied by the current L.P. solution since $x_j = 0$ for $j \notin I$ and so (15.4) is a cut.

Statement of the Algorithm

The initial continuous problem solved by the algorithm is the L.P. problem obtained by ingoring integrality.

Step 1

Solve current continuous problem.

Step 2

If the solution is integral it is the optimal integer solution, otherwise.

Step 3

Choose a basic variable x_i which is currently non-integer. construct the corresponding constraint 15.4 and add it to the problem. Go to step 1.

We note that the tableau obtained after adding the cut is dual feasible and so the dual simplex algorithm is used to re-optimise.

Example.

(15.5) Maximize $x_1 + 4x_2$

Subject to

$$2x_1 + 4x_2 \leq 7$$
$$10x_1 + 3x_2 \leq 14$$
$$x_1, x_2 \geq 0$$

B.V.	x_1	x_2	x_3	x_4	ξ_1	ξ_2	R.H.S.
x_0	-1	-4					0
x_3	2	$\boxed{4}$	1				7
x_4	10	3		1			14
~							
x_0	1		1				7
x_2	$\frac{1}{2}$	1	$\frac{1}{4}$				$7/4$ *
x_4	$\frac{17}{2}$		$-\frac{3}{4}$	1			$35/4$
ξ_1	$\boxed{-\frac{1}{2}}$		$-\frac{1}{4}$		1		$-3/4$
x_0			$\frac{1}{2}$		2		$11/2$
x_2		1			1		1
x_4			$\boxed{-5}$	1	17		-4
x_1	1		$\frac{1}{2}$		-2		$3/2$
~							
x_0				$1/10$	$37/10$		$51/10$ *
x_2		1			1		1
x_3			1	$-\frac{1}{5}$	$-17/5$		$4/5$
x_1	1			$\frac{1}{10}$	$-3/10$		$11/10$
ξ_2				$\boxed{-\frac{1}{10}}$	$-7/10$	1	$-1/10$
x_0					3		5
x_2		1			1		1
x_3			1		-1	-2	1
x_1	1				-1	1	1
x_4				1	7	-10	

	x_1	x_2	x_3	x_4	x_5	x_6	R
	-1	-4					0
	2	$\boxed{4}$	1				7
	10	3		1			14

↓ 'continuous relaxation' solved by one pivot, solution non-integer, add a cut.

	x_1	x_2	x_3	x_4	x_5	x_6	R
x_0	1		1				7
x_2	$\frac{1}{2}$	1	$\frac{1}{4}$				$\frac{7}{4}$*
x_4	$\frac{17}{2}$		$-\frac{3}{4}$	1			$\frac{35}{4}$
ξ (a)	$\frac{1}{2}$		$\boxed{\frac{1}{4}}$ (b)		-1		$\frac{3}{4}$ ↙

↑ current solution infeasible, solve a phase 1 problem.

	x_1	x_2	x_3	x_4	x_5	x_6	R
x_0	-1				4		4
x_2		1			1		1
x_4	$\boxed{10}$				-3		11
x_3	2		1		-4		3

↓ 2nd 'continuous' problem solved, but non-integer, add a cut.

	x_1	x_2	x_3	x_4	x_5	x_6	R
x_0				$\frac{1}{10}$	$\frac{37}{10}$		$\frac{51}{10}$*
x_2		1			1		1
x_1	1			$\frac{1}{10}$	$-\frac{3}{10}$		$\frac{11}{10}$
x_3			1	$-\frac{1}{5}$	$-\frac{17}{5}$		$\frac{4}{5}$
ξ (c)				$\boxed{\frac{1}{10}}$	$\frac{7}{10}$	-1	$\frac{1}{10}$

↑ current solution infeasible, solve a phase 1 problem.

	x_1	x_2	x_3	x_4	x_5	x_6	R
x_0					3		5
x_2		1			1		1
x_1	1				-1	1	1
x_3			1		-1	-2	1
x_4					7	-10	1

(a) Cut generated is $\frac{1}{2}x_1 + \frac{1}{4}x_3 \geq \frac{3}{4}$ or $\frac{1}{2}x_1 + \frac{1}{4}x_3 - x_5 = \frac{3}{4}$

(Add artificial ξ to this, in order to get a basic feasible solution to an augmented set of equations, gives $\frac{1}{2}x_1 + \frac{1}{4}x_3 - x_5 + \xi = \frac{3}{4}$.

(b) Choose a pivot that will reduce ξ (pivoting in col. 1 is allowable, but the arithmetic is worse).

(c) Cut generated is $\frac{1}{10}x_4 + \frac{7}{10}x_5 - x_6 = \frac{1}{10}$

One can show that the Gomory cuts $\sum f_j x_j > f$ when expressed in terms of the original non-basic variables have the form $\sum w_j x_j \leq W$ where the w_j, W are integer and the value of $\sum w_j x_j$ after solving the current continuous problem is $W + \varepsilon$ where $0 < \varepsilon < 1$ assuming the current solution non-integer. Thus the cut is obtained by moving a hyperplane parallel to itself to an extend which cannot exclude an integer solution. It is worth noting that the plane can usually be moved further without excluding integer points thus generating <u>deeper</u> cuts. For a discussion on how this can be done see the reference given for integer programming.

Further Remarks

1) After adding a cut and carrying out one iteration of the dual simplexalgorithm the slack variable corresponding to this cut becomes nonbasic. If during a succeeding iteration this slack variable becomes basic then it may be discarded along with its current row without affecting termination. This means that the tableau never has more than $n + 1$ rows or $m + n$ columns.

2) A valid cut can be generated from any row containing a non-integral variable. One strategy is to choose the variable with the largest fractional part as this helps' to produce a 'large' change in the objective value. It is interesting that finitness of the algorithm has not been proved for this strategy although finitness has been proved for the strategy of always choosing the 'topmost' row the tableau with a non-integer variable.

3) The behaviour of this algorithm has been erratic. It has for example worked well on set covering problems but in other cases the algorithm has to be terminated because of excessive use of computer time. This raises an important point; if the algorithm is stopped prematurely then one does not have a good sub-optimal solution to use. Thus in some sense the algorithm is unreliable.

It is useful to see what has happened graphically. We first express the cuts in terms of x_1, x_2.

Cut no.1

$$\tfrac{1}{2}\, x_1 + \tfrac{1}{4} x_3 \geq \tfrac{3}{4}$$

since

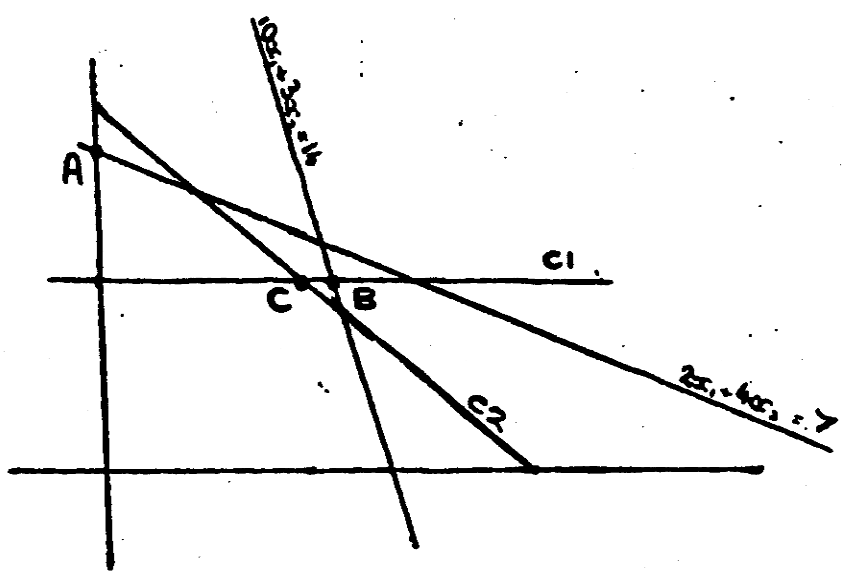
$x_3 = 7 - 2x_1 - 4x_2$ this becomes

$x_2 \leq 1.$

Cut no.2

After re-arranging, this becomes

$$x_1 + x_2 \leq 2$$



A is optimal solution ignoring integrality

B is optimal solution after adding cut C1

C is optimal integer solution found after adding C2.

Branch and Bound Method

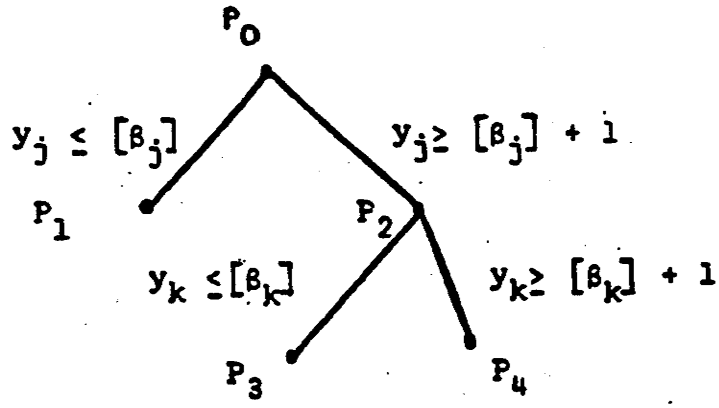
The method to be described in this section constitutes the most successful method applied to date. The idea is quite general and has been applied to many other discrete optimisation problems, e.g. travelling salesman, job shop scheduling.

Let us assume we are trying to solve the mixed integer problem 12.2. Let us call this problem P_0. The first step is to solve the 'continuous' L.P. problem obtained by ignoring the integrality constraints. If in the optimal solution, one or more of the integer variables turn out to be non-integer, we choose one such variable and use it to split the given problem P_0 into two 'sub-problems' P_1 and P_2. Suppose the variable chosen is y_j and it takes the non-integral value β_j in the continuous optimum. Then P_1 and P_2 are defined as follows:

$$P_1 \equiv P_0 \text{ with the added constraint } y_j \leq [\beta_j]$$

$$P_2 \equiv P_0 \text{ with the added constraint } y_j \geq [\beta_j] + 1$$

Now any solution to P_0 is either a solution of P_1 or P_2 and so P_0 can be solved by solving P_1 and P_2. We continue by solving the L.P. problems associated with P_1 and P_2. We then choose one of the problems and if necessary split it into two sub-problems as was done with P_0.



This process can be viewed as the construction of a binary tree of sub-problems whose terminal (pendant) nodes correspond to the problems that remain to be solved.

In an actual computation one keeps a list of the unsolved problems into which the main problem has been split. One also keeps a note of the objective value MIN of the best integer solution found so far.

<u>Step 0</u>

Initially the list consists of the initial problem P_0. Put MIN equal to either the value of some known integer solution, or if one is not given equal to some upper bound calculable from initial data, if neither possibility is possible put MIN = ∞.

Solve the L.P. problem associated with P_0. If the solution has integral values for all integer variables terminate, otherwise

<u>Step 1</u>

Remove a problem P from the list whose optimal continuous objective function value x_0 is less than MIN. If there are no such problems terminate. The best integer solution found so far is optimal. If none have been found the problem is infeasible.

Step 2

Amongst the integer variables in problem P with non-integer values in the optimal continuous solution for P select one for branching. Let this variable be y_p and let its value in the continuous solution be β.

Step 3

Create two new problems P' and P" by adding the extra restrictions $y_p \leq [\beta]$ and $y_p \geq [\beta] + 1$ respectively. Solve the L.P. problems associated with P' and P" and add these problems to the list. If a new and improved integer solution is found store it and update MIN. The new L.P. problems do not have to be solved from scratch but can be re-optimised using the dual algorithm (or parametrically altering the bound on y_p). If during the re-optimisation of either L.P. problem the value of the objective function exceeds MIN this problem may be abandoned. Go to step 1.

If one assumes that each integer variable in P_0 has a finite upper bound (equal to some large number for notionally unbounded variable) then the algorithm must terminate eventually, because as one proceeds further down the tree of problems the bounds on the variables become tighter and tighter, and these would eventually become exact if the L.P. solutions were never integer.

As an example we show a possible tree (Fig 1) for solving

Minimise $20 - 3x_1 - 4x_2$

Subject to

$$\frac{2}{5} x_1 + x_2 \leq 3$$

$$\frac{2}{5} x_1 - \frac{2}{5} x_2 \leq 1$$

$x_1, x_2 \geq 0$ and x_1, x_2 integer

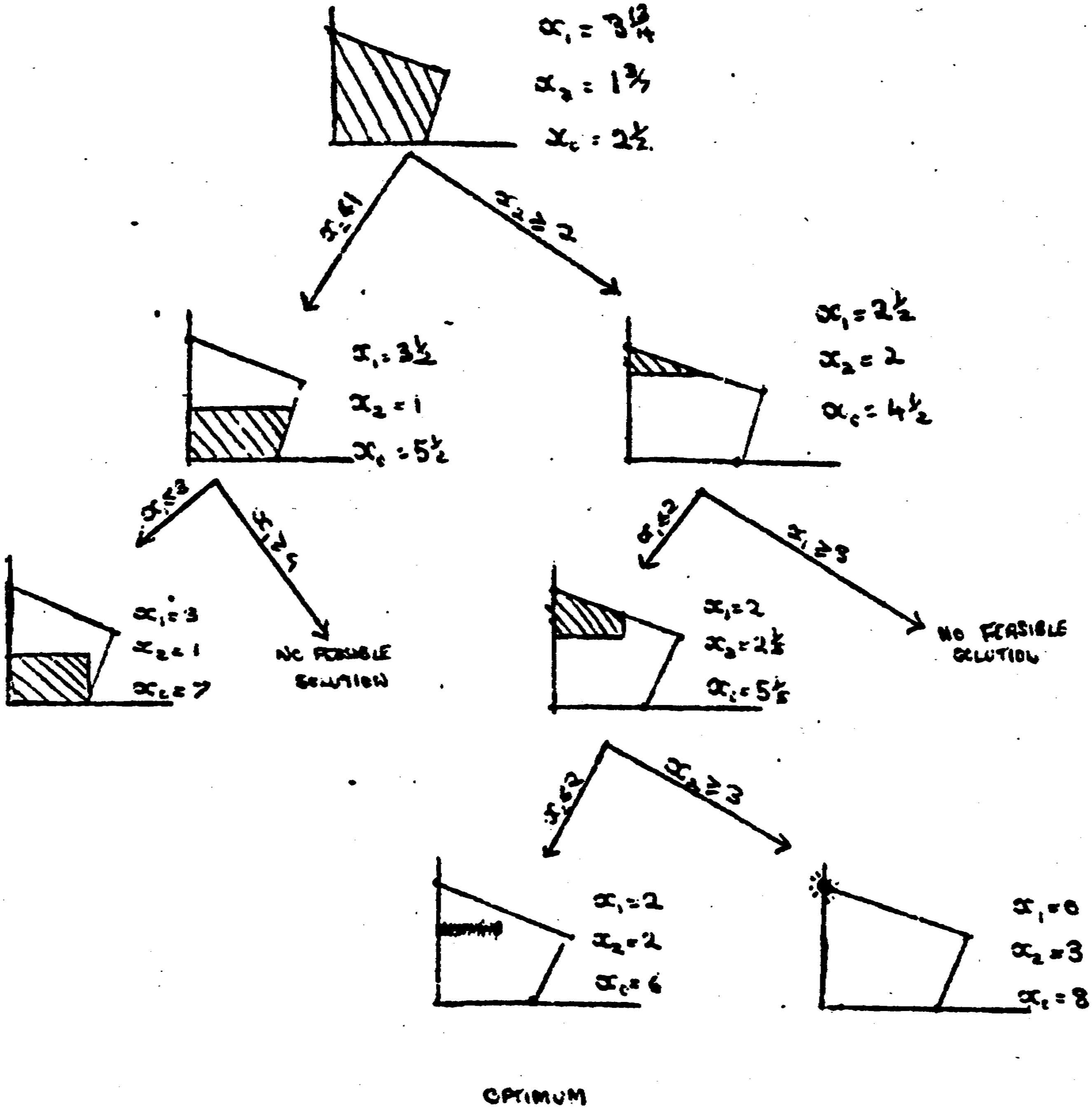

x₁ = 2½
x₂ = 2
x_c = 4½
x₁ = 3½
x₂ = 1
x_c = 5½
x₂ ≥ 2
x₁ = 3
x₂ = 1
x_c = 7
NO FEASIBLE SOLUTION
x₁ ≥ 3
x₁ = 2
x₂ = 2
x_c = 6
x₂ ≥ 3
x₁ = 0
x₂ = 3
x_c = 8
OPTIMUM

Fig 1

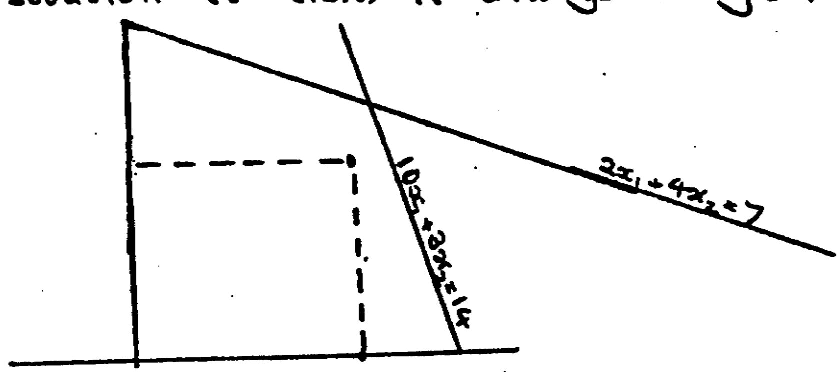
Reformulation of integer programming problems

Consider the problem

$$(15.6) \qquad \text{maximise} \quad x_1 + 4x_2$$
$$\text{subject to} \quad x_1 \leq 1$$
$$x_2 \leq 1$$
$$x_1, x_2 \geq 0 \text{ and integer}$$

It's solution $x_1 =$, $x_2 = 1$ is the same as that of problem (15.5). Indeed for any objective function these 2 problems will have the same solution because they have the same set of integer solutions. – (0,0), (1,0), (0,1), (1,1)

However problem (15.6) is much easier to solve – the continuous solution to (15.6) is always integer.



This illustrates an important point: for a given problem let us denote by IFR the set of integer solutions from which we are searching for the optimum. There are an infinite number of LP problems whose integer solution sets are <u>precisely</u> IFR. For such an LP let CFR(LP) $\supseteq$ IFR denote the set of continuous solutions to the LP.

There will be a unique problem LP^* such that all vertices (basic feasible solutions) of $CFR(LP^*)$ are members of IFR. Thus if we could always identify the corresponding LP^* we could solve it using the simplex algorithm and we would know that its continuous solution would also be the integer solution.

We cannot in general easily identify LP^*. But suppose we have a formulation LP_1 for which the available algorithms are not proving satisfactory, a 'tighter' reformulation to LP_2 where $CFR(LP_1) \supset CFR(LP_2) \supseteq IFR$ can sometimes dramatically improve things.

If you are lucky a reformulation can simply involve adding some extra easily identified constraints satisfied by points in IFR but not by all points in $CFR(LP_1)$.

Implicit Enumeration

Simple B&B algorithm for pure 0-1 problems:

Ex: minimize $z = 7x_1 + 3x_2 + 2x_3 - x_4 - 2x_5$

s.t. $4x_1 + 2x_2 - x_3 + 2x_4 + x_5 \geq 3$

$4x_1 + 2x_2 + 4x_3 - x_4 - 2x_5 \geq 7$

$x_j = 0$ or 1

Top Node: (i) Lower Bound: -3 $\quad x_4 = x_5 = 1$

— NOT FEASIBLE THOUGH $\quad x_1 = x_2 = x_3 = 0$

(ii) Feasibility: $x_1 = x_2 = x_4 = x_5 = 1$ makes (i) feasible

$x_1 = x_2 = x_3 = 1$ makes (ii) feasible.

Branch $x_1 = 1$

or

$x_1 = 0$

$\underline{x_1 = 1}$:

minimize $7 + 3x_2 + 2x_3 - x_4 - 2x_5$

s.t.

$$2x_2 - x_3 + 2x_4 + x_5 \geq -1$$

$$2x_2 + 4x_3 - x_4 - 2x_5 \geq 3$$

LB : 4 NOT FEASIBLE

$x_2 = x_4 = x_5 = 1$ (i) ✓

$x_2 = x_3 = 1$ (ii) ✓

Branch and Bound

September 27, 2018

We consider the problem P_0:

$$\text{Minimize } f(x) \text{ subject to } x \in S_0.$$

Here S_0 is our set of feasible solutions and $f : S_0 \to \mathbb{R}$.

As we proceed in Branch-and-Bound we create a set of sub-problems $\mathcal{P}$. A sub-problem $P \in \mathcal{P}$ is defined by *the description of* a subset $S_P \subseteq S_0$. We also keep a *lower bound* b_P where

$$b_P \leq \min \{ f(x) : \ x \in S_P \} .$$

At all times we act as if we have $x^* \in S_0$, some known feasible solution to P_0 and $v^* = f(x^*)$. If we do not actually have a solution x^* then we let $v^* = -\infty$. We will have a procedure $\textsc{bound}$ that computes b_P for a sub-problem P. In many cases, $\textsc{bound}$ *sometimes* produces a solution $x_P \in S_0$ and sometimes determines that $S_P = \emptyset$.

We initialize $\mathcal{P} = \{P_0\}$.

Branch and Bound:

Step 1 If $\mathcal{P} = \emptyset$ then x^* solves the problem.

Step 2 Choose $P \in \mathcal{P}$. $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\}$.

Step 3 **Bound:** Run $\textsc{bound}(P)$ to compute b_P.

Step 4 If $S_P = \emptyset$ or $b_P \geq v^*$ then we consider P to be solved and go to Step 1.

Step 5 If $\textsc{bound}$ generates $x_P \in S_0$ and $f(x_P) < v^*$ then we update, $x^* \leftarrow x_P, v^* \leftarrow f(x_P)$.

Step 6 **Branch:** Split P into a number of subproblems $Q_i, i = 1, 2, \ldots, \ell$, where $S_P = \bigcup_{i=1}^{\ell} S_{Q_i}$. And $S_{Q_i} \neq S_P$ is a strict subset for $i = 1, 2, \ldots, \ell$.

Step 7 $\mathcal{P} \leftarrow \mathcal{P} \cup \{Q_1, Q_2, \ldots, Q_\ell\}$.

Assuming S_0 is finite, this procedure will eventually terminate with $\mathcal{P} = \emptyset$. This is because the feasible sets S_P are getting smaller and smaller as we branch.

Most often the procedure BOUND has the following form: while it may be difficult to solve P directly, we may be able to find $T_P \supseteq S_P$ such that there is an efficient algorithm that determines whether or not $T_P = \emptyset$ and finds $\xi_P \in T_P$ that minimizes $f(\xi), \xi \in T_P$, if $T_P \neq \emptyset$. In this case, $b_P = f(\xi_P)$ and Step 5 is implemented if $\xi_P \in S_0$. We call the problem of minimizing $f(\xi), \xi \in T_P$, a *relaxed problem*.

Examples:

Ex. 1 Integer Linear Programming. Here S_P is the set of integer solutions and T_P is the set of solutions, if we ignore integrality. The procedure BOUND solves the linear program. If the solution ξ_P is not integral, we choose a variable x, whose value is $\zeta \notin \mathbb{Z}$ and form 2 sub-problems by adding $x \leq \lfloor z \rfloor$ to one and $x \geq \lceil z \rceil$ to the other.

Ex. 2 Traveling Salesperson Person Problem (TSP): Here S_P is the set of tours i.e. single directed cycles that cover all the vertices. We can take T_P to be the set of collections of vertex disjoint directed cycles that cover all the vertices. More precisely, to solve the TSP we must minimise $\sum_{i=1}^n C(I, \pi(i))$ as π ranges over all cyclic permutations. Our relaxation is to minimise $\sum_{i=1}^n C(I, \pi(i))$ as π ranges over all permutations, i.e. the assignment problem. We branch as follows. Suppose that the assignment solution consists of cycles $C_1, C_2, \ldots, C_k, k \geq 2$. Choose a cycle, C_1 say. Suppose that $C_1 = (v_1, v_2, \ldots, v_r)$ as a sequence of vertices. Then in Q_1 we disallow $\pi(v_1) = v_2$, in Q_2 we insist that $\pi(v_1) = v_2$, but that $\pi(v_2) \neq v_3$, in Q_3 we insist that $\pi(v_1) = v_2$, $\pi(v_2) = v_3$, but that $\pi(v_3) \neq v_4$ and so on.

Ex. 3 Implicit Enumeration: Here the problem is

$$\text{Minimize } \sum_{j=1}^n c_j x_j \text{ subject to } \sum_{j=1}^n a_{i,j} x_j \geq b_i,\ i \in [m], \quad x_j \in \{0,1\}, j \in [n].$$

A sub-problem is assciated with two sets $I, O \subseteq [n]$. This the sub-problem $P_{I,O}$ where we add the constraints $x_j = 1, j \in I, x_j = 0, j \in O$. We also check to see if $x_j = 1, j \in I, x_j = 0, j \notin I$ gives an improved feasible solution. As a bound $b_{I,O}$ we use $\sum_{j \notin O} \max\{c_j, 0\}$. To test feasibility we check that $\sum_{j \notin O} \max\{a_{i,j}, 0\} \geq b_i, i \in [m]$. To branch, we split $P_{I,O}$ into $P_{I \cup \{j\}, O}$ and $P_{I, O \cup \{j\}}$ for some $j \notin I \cup O$.