MA423 - Fundamentals of Operations Research
Lecture 4: Integer programming

Katerina Papadaki[1]

Academic year 2017/18

[1]London School of Economics and Political Science. Houghton Street London WC2A 2AE (k.p.papadaki@lse.ac.uk)

Contents

Chapter 1

Integer programming: basic concepts and solution methods

1.1 Introduction

A strong limitation in the applicability of linear programming is that the variables can take arbitrarily fractional values in an optimal solution. In most practical situations however, many quantities are only allowed to take discrete values. For example, it usually does not make sense to return a fractional answer, if a company has to decide about the number of storage facilities to open, or about the number of vehicles to purchase.

Such problems can be formulated as *integer programs (IPs)*. The problem has several variants. By a *pure integer program*, we mean a linear program with all variables restricted to be integer:

$$\begin{aligned} \max\ & cx \\ \text{s. t. } & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

In practice, we usually have a mixture of continuous and discrete variables. This is called a *mixed integer program*. There are n_1 continuous and n_2 integer variables; the vector of variables, the objective vector and the matrix are partitioned in two parts accordingly: $x = (x_1, x_2)$, $c = (c_1, c_2)$, $A = (A_1, A_2)$.

$$\begin{aligned} \max\ & c_1x_1 + c_2x_2 \\ \text{s. t. } & A_1x_1 + A_2x_2 \leq b \\ & x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{Z}^{n_2} \end{aligned}$$

As an important special case, we can have *binary variables* with only two possible values, 0 and 1. These are also called *indicator variables* as they can model *yes-no* decisions, indicating the choice of certain resources. E.g. $x_s = 1$ expresses the fact that we open a certain storage

facility s, and $x_s = 0$ means that we do not open it.

There is an important tradeoff between linear programming (LP) and integer programming (IP). LP is less powerful for modelling as we cannot enforce discrete values in a solution, that can be done in IPs. On the other hand, there are highly efficient algorithms for LP (such as the Simplex algorithm). There are several algorithms that are not only efficient in practice, but also have theoretical guarantees that they terminate rapidly in terms of the input problem size. In contrast, there are no such guarantees at all for integer programming (it is a 'hard' problem in a rigorous mathematical sense of hardness). Whereas several algorithms perform reasonably well in practice, they are slower than linear programming algorithms often by orders of magnitudes, and moreover, we do not have theoretical guarantees that they terminate in a reasonable amount of time.

The most fundamental techniques for solving integer programs are Branch-and-Bound and the Cutting Plane Method; we give an overview of the Branch-and-Bound method in this lecture. Integer programming is implemented in many commercial solvers, that are able to solve typical problems of remarkable size efficiently (but much slower than LPs). Linear programming can be thought of as a swiss army knife due to its simplicity and efficiency. While formulating models is also simple for integer programming, the solution methods are far from simple. They are built on decades of extensive research and expertise. Rather than a swiss army knife, an IP solver could be compared to a contractor with a large toolbox and years of practical experience - yet we might still not be completely sure that he is able to repair our household problem. Moreover, his work is much more expensive than to fix it ourselves.

For this reason, we should always try to use as few integer variables as necessary: use them only if it is really needed and try to keep their number and usage limited.

1.2 Solving Integer Programs

In this section, we describe the fundamental Branch and Bound method for solving integer programs.

Let us start with a simple example.

$$\begin{aligned} \max\ & 2x_1 + 5x_2 \\ & 12x_1 + 5x_2 \leq 60 \\ & 2x_1 + 10x_2 \leq 35 \\ & x_1, x_2 \geq 0 \quad \text{and integer} \end{aligned}$$

The feasible region is illustrated in Figure 1.1 below, where the dots are the feasible integer points. The linear program optimum is at the point $x_1 = 3.864$, $x_2 = 2.727$ and the objective function $2x_1 + 5x_2$ is 21.364. Visually we can see that the possible candidate optimum points are $(2, 3)$ with objective value 19, and $(4, 2)$ with objective value 18 - hence the optimal solution is $x_1 = 2$, $x_2 = 3$.

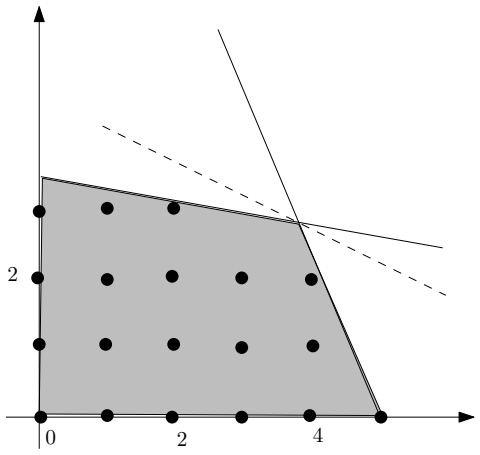

Figure 1.1

For LPs, the solution methods were based on the fact that every local optimal solution is also a global optimum. This is also not true for integer programs. For example, the point $(4, 2)$ does not have any integer neighbours that would be feasible and also give a better objective value - yet it is not optimal. In the LP case we used dual values to verify optimality but here we cannot do that. There is no simple certificate to convince yourself that you have found an optimal solution.

1.2.1 Linear programming relaxation

The difference between an LP problem and an IP problem consists of the fact that the IP has additional constraints, namely the integrality constraints. A problem B that is derived from a problem A by not taking into account some of the integrality constraints is called a *relaxation* of this problem. When we disregard the integrality constraints of an IP problem, we arrive at a normal LP problem. Therefore, such an LP problem is called the LP relaxation of the original IP problem. As the LP relaxation has a larger feasible region than the original IP, the objective function value of an IP can never be better than that of the LP relaxation of the problem. In other words: For a maximization (minimization) problem, the LP relaxation of an IP problem provides an upper (lower) bound for the objective function value of the optimal solution of the IP problem. On Figure 1.1, the gray region is the LP relaxation, and 21.364 is the optimal value of the relaxed problem, which is an upper bound to the integer optimal value of 19.

Given the fractional optimal solution, we might try rounding it to the closest integer solution. However, this might be infeasible, or feasible but not optimal. In the above example, the fractional optimal solution was $(3.864, 2.727)$. Rounding it would give $(4, 3)$, that is infeasible as in the second constraint we get $38 \leq 35$. Even if we round it down to get $(3, 2)$, which is feasible, the solution has objective value 16, which is far from 19. So even if the rounded solution is feasible, it might be far from optimal.

Yet sometimes we might get an acceptable solution by rounding. This is possible if

- The rounded solution is feasible, *and*

- The difference between the optimal objective value of the LP relaxation and the objective function value of the rounded feasible solution is acceptable for the practical purpose that the model serves.

If the variables are binary then $(\frac{1}{3}, \frac{1}{2})$ could round to $(0, 1)$ (if feasible) and the difference in objective values could be huge. Also, when rounding binary variables in many dimensions then rounding to a feasible solution could prove to be really hard.

1.2.2 The Branch and Bound algorithm

A lot of research has been and is being applied to methods to solve integer programming problems. The most successful approach to date the is *branch and bound*, developed in 1960 by Ailsa Land and Alison Doig, at the LSE Operational Research Department (the predecessor of the Operations Research Group). All the commercial mathematical programming systems offer an integer programming solution procedure based on some variation of this method.

Branch and bound can be thought of as a systematic exploration of the feasible region and an elimination of those parts which can be shown either not to contain an integer solution or not to contain the optimal solution.

The branch and bound algorithm is a divide-and-conquer approach: the feasible region is partitioned into a collection of smaller regions. Each region is itself represented by linear constraints and the objective function can be optimised over these smaller regions. The partitions of the feasible region can themselves be further partitioned into even smaller regions. In practice the partitioning of the feasible region is performed sequentially. The algorithm is best represented by a tree graph.

An example of the Branch and Bound algorithm

Let us return to the example we considered earlier. The algorithm starts by solving the LP relaxation. This corresponds to the *root node* of the *branch and bound tree*. If the solution satisfies the integer condition we stop (this is the optimal integer solution), otherwise it is necessary to branch. Select a variable whose value is non-integer and create two new sub-problems. The constraints of the new sub-problems are chosen so that the current non-integer solution is infeasible in both sub-problems.

In the example, the solution at the root node is: $x_1 = 3.864$, $x_2 = 2.727$ with objective value 21.363. This objective value is an upper bound on the value of the optimal integer solution x^*: $z(x^*) \leq 21.363$. In fact, since the objective coefficients of the IP are integer we know that the optimal objective value is integer and thus we can round down 21.363 to 21 and get the upper bound: $z(x^*) \leq 21$. This step is called the *bounding step*.

We arbitrarily choose to branch on x_1, since x_1 is fractional (non-integer). This is done by creating two sub-problems: one with the extra constraint $x_1 \leq 3$ and the other with $x_1 \geq 4$. (Figure 1.2(a)). The two subproblems are shown on Figure 1.2(b). This step is called the *branching step*.

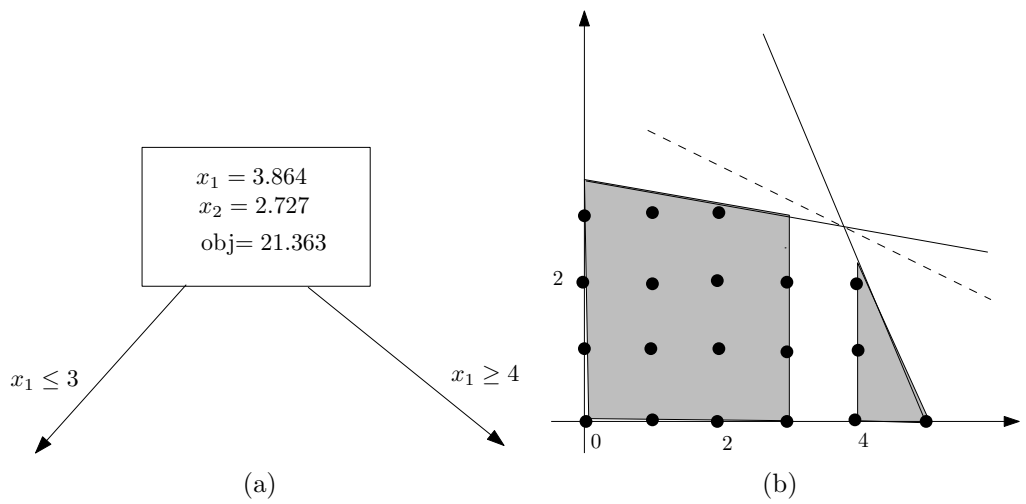

Figure 1.2

We choose (arbitrarily) to solve the left-hand problem first. This gives a solution $x_1 = 3$ and $x_2 = 2.9$ with objective value 20.5. Notice that because an additional constraint, $x_1 \leq 3$, has been added, the value of the objective function either stays the same or decreases. This gives a tighter upper bound of 20 for the integer solution of the current sub-problem (after rounding down from 20.5).

As x_2 is non-integer two further sub-problems with the constraints $x_2 \leq 2$ and $x_2 \geq 3$ are created (Figure 1.3). There are now three unsolved sub-problems.

If we now solve the leftmost problem ($x_1 \leq 3$, $x_2 \leq 2$) this gives an integer solution $x_1 = 3$, $x_2 = 2$ with objective value 16. We call this solution an *incumbent* solution, which means the best known integer solution so far. This integer solution provides a lower bound on the original problem $x^* : 16 \leq x^* \leq 21$. So now we know that the optimal IP value is an integer between 16 and 21. Note that whereas the LP relaxation gives an upper bound, finding feasible integer solutions give lower bounds. For this sub-problem we found the best integer solution so we do not branch any further. This branch or subproblem ($x_1 \leq 3$, $x_2 \leq 2$) is said to be *fathomed (conquered) by integrality.*

Now, the algorithm backtracks and selects one of the unsolved sub-problems. Arbitrarily choosing the most recently formed unsolved sub-problem ($x_1 \leq 3$, $x_2 \geq 3$)and solving it gives the tree on Figure 1.4. The objective value is 20.

Moving to sub-problem with constraints $x_1 \leq 3$, $x_2 \geq 3$, and $x_1 \leq 2$ is equivalent to solving the sub-problem with constraints $x_1 \leq 2$, $x_2 \geq 3$. This produces solution $x_1 = 2$, $x_2 = 3.1$, with objective value 19.5. Thus, the bounds for the optimal integer solutions on this subproblem are: $16 \leq x^* \leq 19$. From here we branch with respect to variable x_2 by adding constraints: $x_2 \leq 3$ and $x_2 \geq 4$. Solving the relaxation of the left subproblem with $x_2 \leq 3$, gives an integer solution $x_1 = 2$, $x_2 = 3$ with objective value 19. This is now the incumbent solution and we say that this sub-problem is *fathomed by incumbent solution.* So we do not branch on this sub-problem. Now if solve the relaxation of the right subproblem with $x_2 \geq 4$ we get an infeasible solution. Thus we will not be able to find any more solutions by branching further so we do not branch

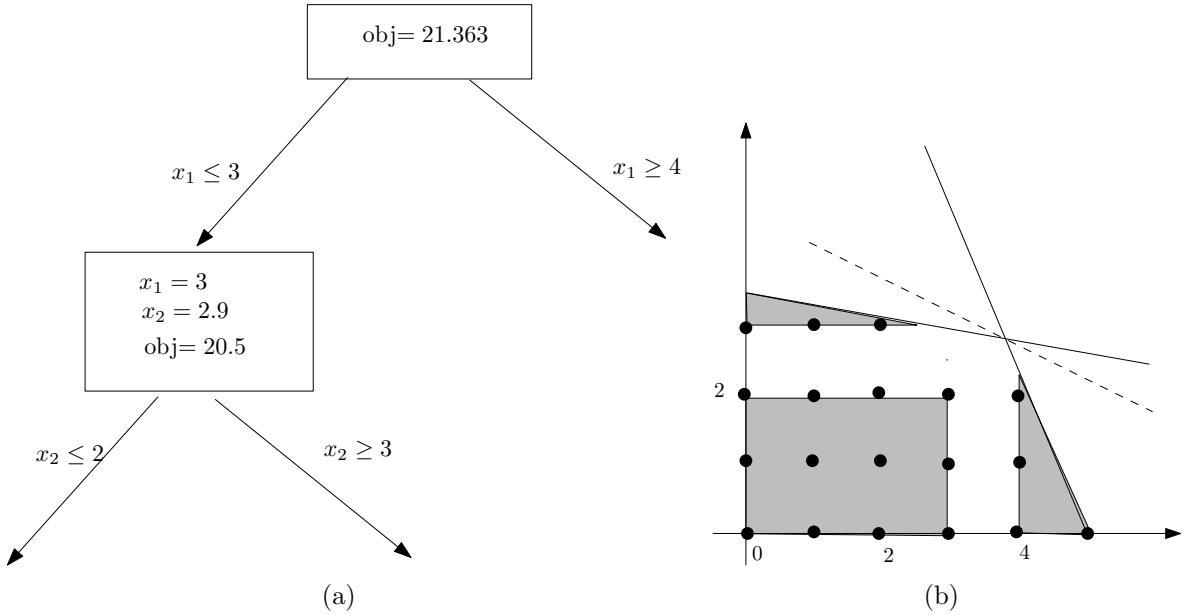

Figure 1.3

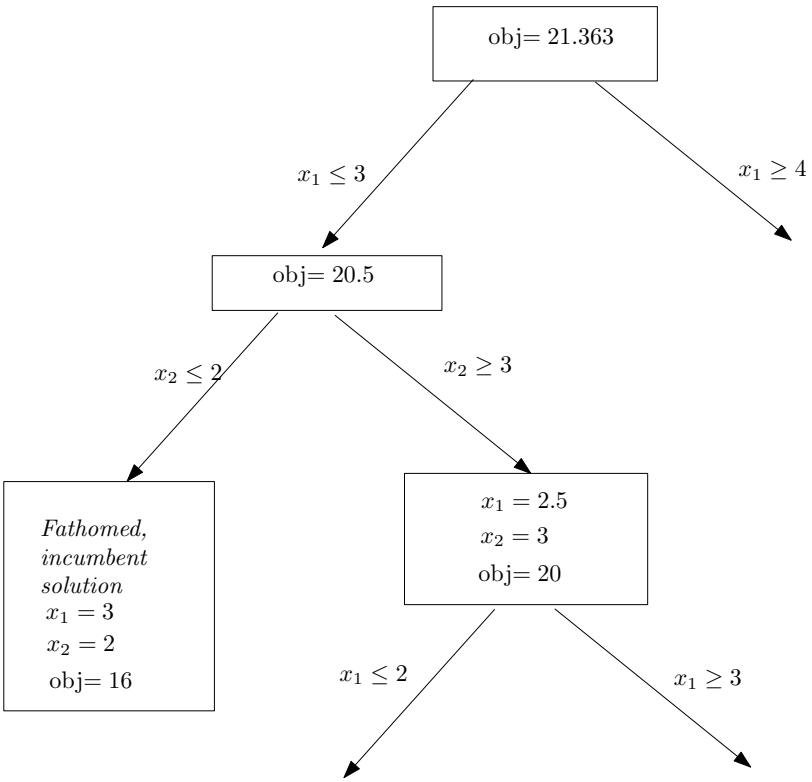

Figure 1.4

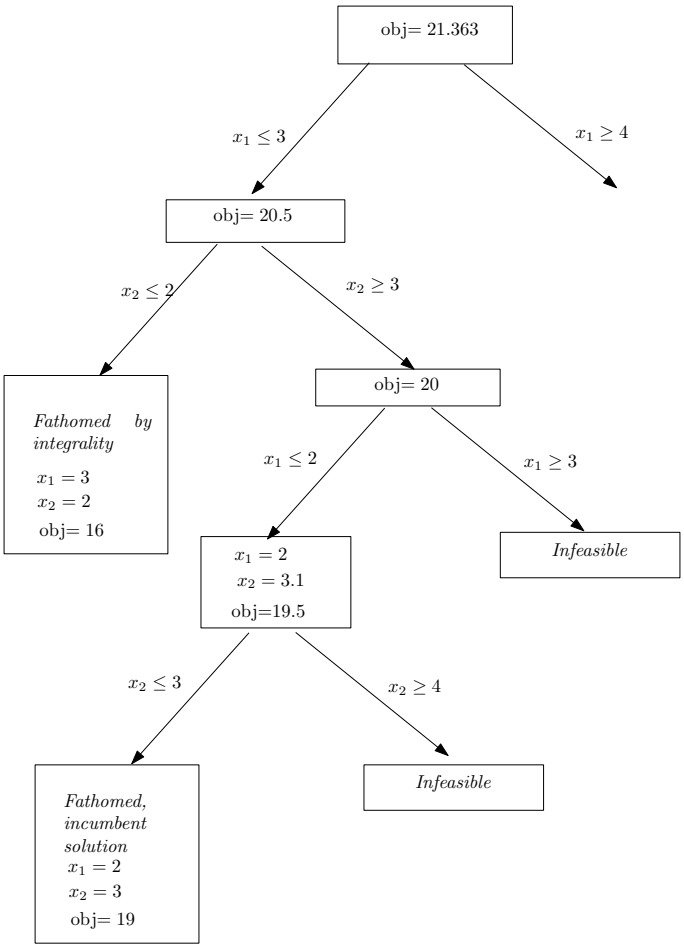
obj= 21.363
$x_1 \leq 3$
$x_1 \geq 4$
obj= 20.5
$x_2 \leq 2$
$x_2 \geq 3$
Fathomed by integrality
$x_1 = 3$
$x_2 = 2$
obj= 16
obj= 20
$x_1 \leq 2$
$x_1 \geq 3$
$x_1 = 2$
$x_2 = 3.1$
obj=19.5
Infeasible
$x_2 \leq 3$
$x_2 \geq 4$
Fathomed, incumbent solution
$x_1 = 2$
$x_2 = 3$
obj= 19
Infeasible

Figure 1.5

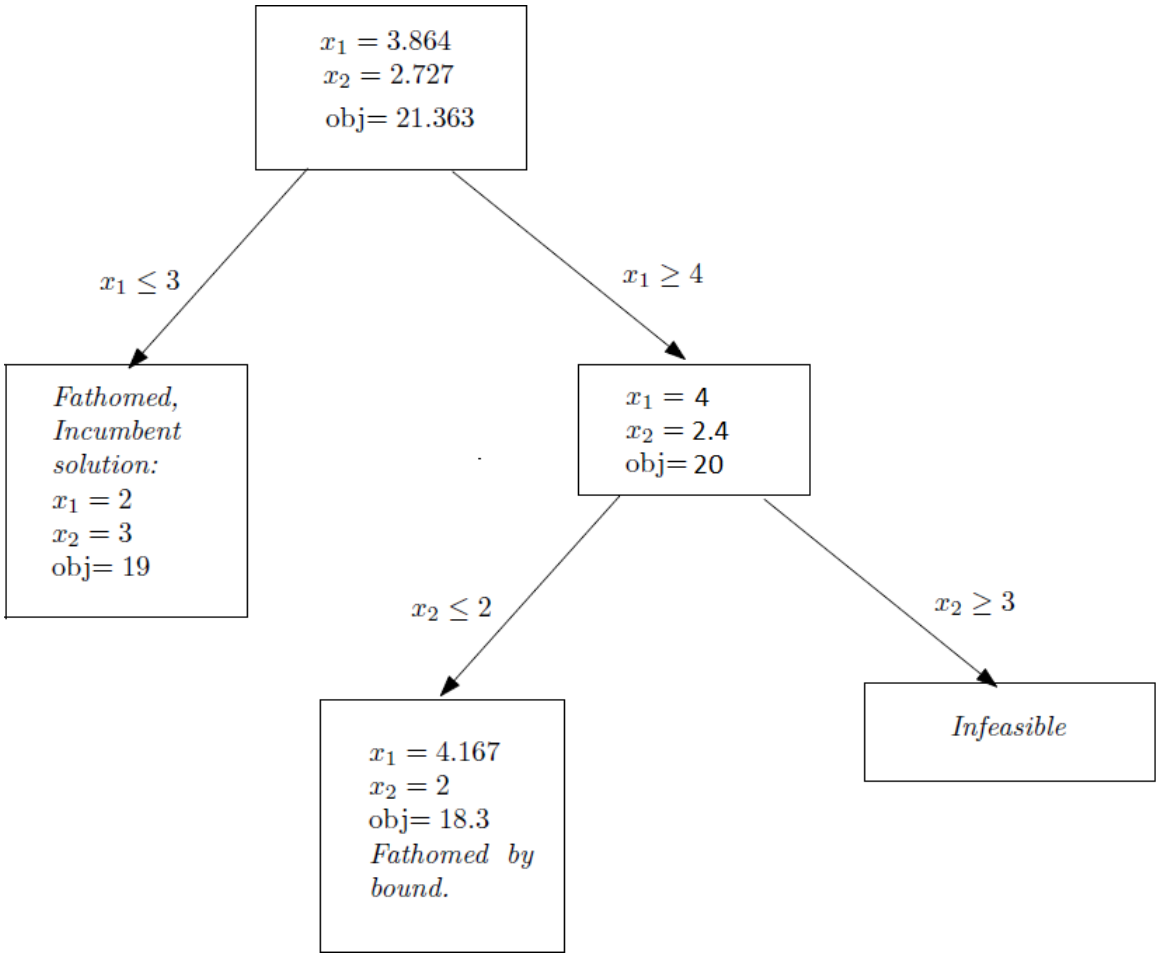
$x_1 = 3.864$
$x_2 = 2.727$
obj= 21.363
$x_1 \leq 3$
$x_1 \geq 4$
Fathomed, Incumbent solution:
$x_1 = 2$
$x_2 = 3$
obj= 19
$x_1 = 4$
$x_2 = 2.4$
obj= 20
$x_2 \leq 2$
$x_2 \geq 3$
$x_1 = 4.167$
$x_2 = 2$
obj= 18.3
Fathomed by bound.
Infeasible

Figure 1.6

on this sub-problem. We say that this branch is *fathomed by infeasibility*.

Continuing we solve the relaxation of sub-problem $x_1 \leq 3$, $x_2 \geq 3$, and $x_1 \geq 3$, which is equivalent to adding the constraints $x_1 = 3$ and $x_2 \geq 3$, and we find that the problem is infeasible. Thus we ignore this sub-problem and do not branch on it.

We have now considered all options that branch from sub-problem $x_1 \leq 3$. The resulting sub-tree is shown in Figure 1.5. The incumbent solution at this stage is $(2, 3)$ with objective value 19 and the bounds are: $19 \leq x^* \leq 21$. This means that if the optimal integer solution satisfies $x_1 \leq 3$ then it must be $(2, 3)$.

At this point, the entire branch of the tree where $x_1 \leq 3$ is completely fathomed. So we backtrack to the very beginning and solve the sub-problem with the single extra constraint $x_1 \geq 4$. Continuing in the same fashion as before and subsequently branching gives the tree on Figure 1.6. At the sub-problem with $x_1 \geq 4$, $x_2 \leq 2$, the LP optimum value is 18.3. As extra constraints are added to form the sub-problems, the value of the objective function cannot increase, it either stays the same of decreases. Consequently, in all later sub-problems on this branch we get solutions with objective value less than or equal to 18.3. However we have a better incumbent solution of value 19. Consequently, there is no point in continuing to branch. We say that the sub-problem $x_1 \geq 4$, $x_2 \leq 2$ is *fathomed by bound*. The sub-problem $x_1 \geq 4$, $x_2 \geq 3$ gives an infeasible solution, here we can say that this branch is *fathomed by infeasibility*. Thus no better solution has come out of branch $x_1 \geq 4$.

The incumbent, and thus optimal solution to the problem at the end of the entire branch and bound search is $x_1 = 2$, $x_2 = 3$ with an objective function value of 19.

In the branch and bound algorithm there are three ways that a sub-problem can be **fathomed**, which means conquered or dismissed from consideration:

1. *Fathomed by integer solution*: When the solution of the LP relaxation of the sub-problem is integer then there is no further need to branch since we have solved optimally the IP sub-problem. If this integer solution is better than the current incumbent solution then it becomes the new incumbent and we say that it is *fathomed by the incumbent solution*.

2. *Fathomed by infeasibility*: When the LP relaxation of the sub-problem has no feasible solutions then we know that the IP sub-problem is also infeasible and thus we do not branch any further.

3. *Fathomed by bound*: When the solution of the LP relaxation of the IP sub-problem has objective value (or rounded objective value if objective coefficients are integer) that is less than the objective value of the incumbent solution then we know that if we branch further we will only get worse integer solutions and thus we do not branch any further.

Branch and bound algorithm

- **Initialize**: Apply the bounding step, fathoming step and optimality test to the whole problem. If not fathomed then classify this problem as one remaining sub-problem and perform the iteration steps below:

- Iteration steps:

    1. **Branching step**: amongst the remaining unfathomed problems select one. Branch on one of the variables that did not have integer value in the LP relaxation.
    2. **Bounding step**: for each new subproblem apply the simplex algorithm to its LP relaxation to obtain an optimal solution and an objective value. If the objective coefficients are integer then round down (up for minimising) this value. This rounded objective value is an upper bound (lower bound when minimising) on the objective value of the IP sub-problem.
    3. **Fathoming step**: For each sub-problem apply the three fathoming tests summarised above and discard all the sub-problems that are fathomed by any of the test.

- **Optimality check**: Stop when there are no remaining sub-problem. The current incumbent solution is optimal.

Remarks:

1. We do not specify above how to pick the next sub-problem. You can pick the one with the highest bound (lowest when minimizing) because this sub-problem would be the most promising one to contain an optimal solution to the whole problem. Or you could pick the one that was created most recently. This is because this is the most similar to the one you just solved by simplex and simplex could use re-optimisation techniques to solve it faster.

2. If your objective coefficients are not integer then you should not round the bound in the bounding step.

3. If your variables were binary integer variables then for branching variable x_1 your branches would be equalities: $x_1 = 1$ and $x_1 = 0$.

4. Notice that in order to solve this problem in 2 variables it was necessary to solve 11 linear programs. In general, the number of steps may blow up exponentially: if we have k binary variables, we might end up with 2^k different cases. It requires substantially more work to solve an integer program than a linear program - in some cases, it takes prohibitively long. That is why great care should be taken in setting up an integer program and check whether there is possibly a way to formulate the problem in a more economic way, with fewer integer variables.

1.3 Mixed integer programming

As we discussed earlier mixed integer programs (MIPs) contain variables that are both integer and continuous. You can apply the branch and bound algorithm the same way as described above for MIPs with the following simple changes:

1. Branching step: You only branch on variables that are integer.

2. Bounding step: You do not round down (up for minimisation) the bound of the LP relaxation since the objective value of the MIP is most likely fractional.

3. Fathoming step: A solution is considered incumbent if it is integer in the integer variables.

Chapter 2

Integer programming: Formulations using binary variables

We illustrate typical usage of binary variables in modelling by giving a number of examples.

2.1 Fixed costs - the big-M method

Let z represent the quantity of product to be manufactured. The cost per unit of production is c. However, this is just the variable production cost. If any of the product is manufactured a fixed cost is incurred which is f. That is:

$$\text{production cost} = \begin{cases} 0 & \text{if } z = 0, \\ f + cz & \text{if } z > 0. \end{cases}$$

Whereas this cost function might seem linear, it is *not*. (Setting the cost to be $f+cz$ everywhere would be linear, however, it would give f for $z = 0$ instead of 0.) We will need a binary variable to express such a cost function.

We wish to minimise the total cost. To model this situation we introduce a new binary $(0-1)$ variable δ, which we will use to represent the logical states: *(a)* no product is produced; *(b)* some amount $z > 0$ is produced. In other words we need to model the following situation:

$$\text{If } z > 0 \text{ then } \delta = 1. \tag{2.1}$$

Equivalently, this can be written as:

$$\text{If } \delta = 0 \text{ then } z = 0. \tag{2.2}$$

To model this logical condition it is necessary to introduce the maximum amount of the product that can be produced, that is the maximum value of z. Let this be M (we do not need the exact maximum value, only an upper bound). We now model this situation by introducing the

following constraint:

$$z \leq M\delta. \tag{2.3}$$

Clearly, if $z > 0$ (i.e. some production is undertaken), then δ is forced to 1 ($z > 0$ and $\delta = 0$ would give the contradiction $0 < z \leq 0$). Thus we can now write the total cost (to be minimised) as

$$\text{production cost} = f\delta + cz.$$

Note that because we are minimising cost, the objective will be trying to set $\delta = 0$ if this is possible, so that we don't have to worry about fixed cost being incurred when no production is undertaken, i.e. when $z = 0$ then the minimisation will set $\delta = 0$.

This is a widely applicable modelling strategy, often referred to as the big-M method. We can introduce a binary variable δ to model the relationship (2.1), provided an upper bound M on the possible values of z. Note that the inequality (2.3) does not directly imply that $\delta = 0$ whenever $z = 0$. This is achieved only because we want to minimise $f\delta$ in the objective.

Sometimes we may need to formulate this converse implication directly, as illustrated by the next example.

2.2 A blending problem - the small-m method

We have to blend three ingredients, A, B and C. The extra condition is: if A is included in the blend then B must also be included. Notice that the blend could be made entirely of ingredient B. Let x_A, x_B and x_C denote the proportions of the ingredients. The maximum value of all three variables is 1 (that is 100%). The minimum proportion at which B is considered to be present in the blend is $m = 0.01$.

We may model this situation by introducing a binary variable δ indicating whether A is in the blend or not. The situation is now modelled as follows.

$$\begin{aligned} x_A &\leq \delta \\ x_B &\geq m\delta \\ x_A + x_B + x_C &= 1 \\ x_A, x_B, x_C &\geq 0 \\ \delta &\in \{0, 1\} \end{aligned}$$

If $\delta = 0$ this forces $x_A = 0$, i.e. the blend will be made entirely of products B and C. On the other hand if $\delta = 1$, this forces $x_B \geq m$, i.e. the proportion of B will be at least $m = 0.01$.

Such a formulation is called the small-m method. For a variable z ($z = x_B$ in the example), we aim to express the converse of (2.1), that is

$$\text{If } \delta = 1 \text{ then } z > 0. \tag{2.4}$$

or equivalently,

$$\text{If } z = 0 \text{ then } \delta = 0. \tag{2.5}$$

Analogously to the upper bound M in the previous method, we need a lower bound $m > 0$ which expresses the intended minimal positive value of z. Assuming that we always have $z \geq m$ we can use the inequality:

$$z \geq m\delta.$$

which guarantees:

$$\text{If } \delta = 1 \text{ then } z \geq m \text{ and thus } z > 0. \tag{2.6}$$

Thus, the variable z can only take value 0 or greater than or equal to m. Values of z in $0 \leq z < m$ are neglected.

2.3 Facility location

Let us illustrate the "big-M-method" on a more complex modelling example. A logistic company has to supply a set T of stores from a set F of possible facilities. They can choose an arbitrary set of these facilities to open. For each facility f, there is a significant opening cost b_f they have to pay irrespective to how intensively this facility will be used.

The other part of the cost is transportation. Each store t has to be supplied with r_t units of goods, and the cost of transportation is different between different pairs of facilities and stores: the cost of transporting one unit of goods from facility f to store t is c_{ft}. We must of course supply stores from open facilities.

Let us assign a binary variable δ_f to each facility: $\delta_f = 1$ means this facility is open and $\delta_f = 0$ means it is not. Let the variable x_{ft} express the amount of goods supplied by facility f to store t. A natural upper bound on x_{ft} is r_t as that is the total amount of goods needed in store t. We have the following big-M-type constraint ensuring that we only use open facilities:

$$x_{ft} \leq r_t \delta_f \quad \text{for all } f \in F, t \in T.$$

The entire model can be formulated as follows:

$$\begin{aligned}
\min \sum_{f \in F} b_f \delta_f + \sum_{t \in T} \sum_{f \in F} c_{ft} x_{ft} & \\
\sum_{f \in F} x_{ft} = r_t & \qquad \text{for all } t \in T, \\
x_{ft} \leq r_t \delta_f & \qquad \text{for all } f \in F, t \in T, \\
x_{ft} \geq 0 & \qquad \text{for all } f \in F, t \in T, \\
\delta_f \in \{0, 1\} & \qquad \text{for all } f \in F.
\end{aligned}$$

Let us now include an additional restriction: every store must choose a facility and has to get its entire supply from that: it is not possible to transport goods from multiple facilities.

This constraint can be included by adding several additional binary variables. Let β_{ft} express that store t is supplied from facility f. First, we need that stores can be only supplied from open facilities. This means that $\beta_{ft} = 1$ should not be allowed if $\delta_f = 0$. This can be added by the simple inequality

$$\beta_{ft} \leq \delta_f \quad \text{for all } f \in F, t \in T.$$

Instead of the previous big-M-type constraints, we must formulate that

$$x_{ft} = \begin{cases} r_t & \text{if } \beta_{ft} = 1 \\ 0 & \text{if } \beta_{ft} = 0. \end{cases}$$

This can be captured by the following constraint.

$$x_{ft} = r_t \beta_{ft} \quad \text{for all } f \in F, t \in T.$$

Finally, we need to make sure that for every store $t \in T$, we will choose to supply it from exactly 1 facility. That is, among the values β_{ft} over all $f \in F$, we must have exactly one 1 and all others 0. This is enforced by

$$\sum_{f \in F} \beta_{ft} = 1 \quad \text{for all } t \in T. \tag{2.7}$$

Let us summarise the model

$$\begin{aligned} \min \sum_{f \in F} b_f \delta_f + \sum_{t \in T} \sum_{f \in F} c_{ft} x_{ft} & \\ \beta_{ft} \leq \delta_f & \qquad \text{for all } f \in F, t \in T, \\ \sum_{f \in F} \beta_{ft} = 1 & \qquad \text{for all } t \in T \\ x_{ft} = r_t \beta_{ft} & \qquad \text{for all } f \in F, t \in T, \\ x_{ft} \geq 0 & \qquad \text{for all } f \in F, t \in T, \\ \delta_f \in \{0, 1\} & \qquad \text{for all } f \in F, \\ \beta_{ft} \in \{0, 1\} & \qquad \text{for all } f \in F, t \in T. \end{aligned}$$

Note that the variables x_{ft} are not really necessary and could be replaced simply by $r_t \beta_{ft}$: all old inequalities on them are implied by the new inequalities on the β_{ft} variables. This would leave us with the variables δ_f and β_{ft} only. Indeed, these do describe the entire solution.

2.4 Expressing logical conditions

We used binary variables in the previous examples to express logical relationships. Let us now systematically explore some cases. Consider two possible events X_1 and X_2. and let us associate

binary variables x_1 and x_2 to them. $x_i = 1$ means that event X_i takes place, and $x_i = 0$ means that it does not. We can express the basic logical operations the following way:

$x_1 + x_2 \geq 1$	'X_1 or X_2'
$x_1 + x_2 = 1$	'either X_1 or X_2'
$x_1 = 1, x_2 = 1$	'X_1 and X_2'
$x_1 = x_2$	'X_1 if and only if X_2'
$x_1 \leq x_2$	'if X_1 then X_2'

Note that 'X_1 or X_2' means that at least one of them or possibly both of the events occur. Whereas 'either X_1 or X_2' means that exactly one of them occurs but not both.

Remarks:

1. Logical conditions often can be transformed into other equivalent logical conditions: 'not (X_1 and X_2)', for example, is equivalent to '(not X_1) or (not X_2)'. Similarly, 'if X_1 then X_2' is equivalent to '(not X_1) or X_2'. We can represent '(not X_1)' by $(1 - x_1)$, thus 'if X_1 then X_2' is represented by $x_1 \leq x_2$, which is equivalent to $(1 - x_1) + x_2 \geq 1$ which is the variable representation of the event '(not X_1) or X_2'.

2. The constraints for the condition 'X_1 and X_2' have been mentioned here because they can be used when an 'and'-condition is part of a larger, more complex logical expression. In the case of a simple 'and'-statement we do not need indicator variables. For constraints stated in LP and IP problems it is assumed that there is an 'and'-relationship between them: we assume the first constraint holds 'and' the second constraint holds 'and' so on. This implies that for an expression like 'X_1 and X_2' it is normally sufficient just to add the expressions X_1 and X_2 as regular constraints.

3. We can generalise this to the case of more than two events and express longer more complicated conditions using binary variables.

Let us now see examples for n events: $X_1, X_2, \ldots, X_n$ with indicator variables $x_1, x_2, \ldots, x_n$. For some $0 \leq k \leq n$ we can use the following inequalities.

$\sum_{i=1}^{n} x_i \leq k$	'at most k of the events can happen'
$\sum_{i=1}^{n} x_i = k$	'exactly k of the events must happen'
$\sum_{i=1}^{n} x_i \geq k$	'at least k of the events can happen'

If you look at constraint (2.7) of the facility location problem, this is an example where the event that every store has to be supplied from exactly one facility was expressed in a constraint

of binary variables.

Example 2.1. As another example, consider the following problem. A clothing company has to decide on the set of products to manufacture. There are four possible types of shirts, A, B, C and D and two types of ties, P and Q. They want to consider manufacturing ties only if they decide to manufacture at least three types of shirts.

If at least one of P and Q is produced, then at least 3 of A, B, C and D must be produced

Let us express this with the corresponding binary variables $x_A, x_B, x_C, x_D, x_P, x_Q$. Let us add one more binary variable y to express that at least one of the ties (P and Q) is being produced. Equivalently, if $x_P + x_Q > 0$, then $y = 1$. This can be done with the big-M method - the maximum value of $x_P + x_Q$ is 2. Hence we can formulate the constraint

$$x_P + x_Q \leq 2y.$$

We then need that if $y = 1$ then $x_A + x_B + x_C + x_D$ is at least 3. We can formulate a constraint analogous to the small-m method, with $m = 3$:

$$3y \leq x_A + x_B + x_C + x_D.$$

The above says that if $y = 1$ then $x_A + x_B + x_C + x_D \geq 3$. These two constraints together express the required relationship.

Remark. It might be tempting to simplify it further by substituting $\frac{1}{2}(x_P + x_Q) \leq y$ from the first inequality, however, it is *not possible*. We would get

$$\frac{3}{2}(x_P + x_Q) \leq x_A + x_B + x_C + x_D. \tag{2.8}$$

Indeed, if exactly one of x_P and x_Q is 1 and the other is 0, the left hand side would be only $\frac{3}{2}$, allowing only 2 of x_A and x_B being 1, that is, producing only 2 shirts. Thus, we need:

$$\frac{3}{2}(x_P + x_Q) \leq 3y \leq x_A + x_B + x_C + x_D. \tag{2.9}$$

Enforcing that y is binary guarantees that when $x_P + x_Q = 1$ we get $y = 1$, which guarantees that the right have side of (2.9) is greater than 3. Without y binary we have (2.8) which fails because it implies that $y = \frac{1}{2}$.

2.5 Non-convex feasible region - Indicator variables for constraints

In the previous example we have seen how binary variables can powerfully model cases when the decision variables may take discrete variables only. Yet we might need to introduce binary

variables even in cases when all decision variables are continuous, but the constraints differ from the usual LP constraints. We consider the following example:

Example 2.2. Assume a small beer brewery has to decide how much lager beer and ale they wish to produce. Whereas they are willing to brew both types, the management has the vision of building a strong brand with a dominant product it can be associated with. Either they want to produce more lager than ale by at least 6,000 barrels, or they want to produce more ale than lager by at least 4,000 barrels. They do not wish to produce anything in between, they want to avoid the two amounts to be roughly the same. Also, the maximum amount they can produce in total is 10,000 barrels. If the amount of lager and ale is x_1 and x_2, then constraint they wish to add is:

$$\text{either } x_1 - x_2 \geq 6000 \tag{2.10a}$$

$$\text{or } x_1 - x_2 \leq -4000. \tag{2.10b}$$

The feasible region cannot be modelled by linear constraints. We need to add a binary variable δ which is 1 if production is "lager-dominant" (2.10a) and 0 if it is "ale-dominant" (2.10b). We aim to keep (2.10a) and make (2.10b) void if $\delta = 1$, and conversely, keep (2.10b) and make (2.10a) void if $\delta = 0$. This will be done similarly to the big-M method.

As the total production is at most 10,000, we get the bounds

$$-10000 \leq x_1 - x_2 \leq 10000.$$

Let us consider the following two inequalities.

$$x_1 - x_2 \geq 6000\delta - 10000(1 - \delta) \tag{2.11a}$$

$$x_1 - x_2 \leq -4000(1 - \delta) + 10000\delta. \tag{2.11b}$$

In case of $\delta = 1$, the first is identical to (2.10a), whereas the second gives

$$x_1 - x_2 \leq 10000.$$

This is void as it is always satisfied due to the constraint on total production. Conversely, if $\delta = 0$, then the first inequality would give the void

$$x_1 - x_2 \geq -10000,$$

whereas the second gives (2.10b). The inequalities (2.11a and b) can be written more concisely as

$$x_1 - x_2 - 16000\delta \geq -10000$$
$$x_1 - x_2 - 14000\delta \leq -4000.$$

In the above example, we formulated dependence of constraints when exactly one of them can occur (either/or relationship), whereas normally we assume that all constraints should occur at the same time (and relationship). We had a linear constraint $ax \leq b$, and a binary variable δ, and we wanted to guarantee that if $\delta = 1$, then $ax \leq b$ is applicable, but otherwise it is not enforced.

Let M denote an upper bound on $ax - b$, i.e. M is the maximum possible violation of this constraint, so the following holds for every solution:

$$ax \leq M \tag{2.12}$$

Then we can formulate the constraint

$$ax \leq b\delta + M(1 - \delta).$$

Indeed, if $\delta = 0$ then this constraint turns into (2.12) that is always true. If $\delta = 1$ then it gives back $ax \leq b$, as required.

We might also need to model the converse relationship: whenever $ax \leq b$ is satisfied, then $\delta = 1$ must hold, or equivalently, $\delta = 0$ implies $ax > b$. $ax > b$ is equivalent to $ax \geq b + m$ for some small $m > 0$.

First we need a lower bound, let $M' \geq 0$ denote a value such that

$$ax \geq -M' \tag{2.13}$$

must always hold. We enforce that if $ax \leq m$ then $\delta = 1$ must hold or equivalenty if $\delta = 0$ then $ax \geq b + m$ for some small $m > 0$ by the constraint:

$$ax \geq (b + m)(1 - \delta) - M'\delta,$$

or more concisely,

$$ax + (b + m + M')\delta \geq b + m.$$

Indeed, if $\delta = 0$, we obtain the required $ax \geq b + m$ for some small $m > 0$. Further, if $\delta = 1$ then this constraint holds trivially due to (2.13) and nothing is enforced.

Thus, if we want to model the relationship: $\delta = 1$ if and only if $ax \leq b$ we need to write:

$$(b + m)(1 - \delta) - M'\delta \leq ax \leq b\delta + M(1 - \delta).$$

2.6 Variables with restricted value ranges

Integer variables are appropriate to use when a variable can only take integer values, e.g. the number of machines of a certain type to be purchased. If our variable x should be an integer

in $\{\ell, \ell+1, \ldots, u-1, u\}$, we can simply declare x as an integer variable with the inequalities $\ell \leq x$, $x \leq u$.

Consider now the additional requirement that x should be divisible by 3. A useful trick is that instead of x, we introduce an integer variable for $\frac{x}{3}$. That is, we declare y to be integer, and add the inequalities $\ell \leq 3y$, $3y \leq u$. In the formulation, we replace each occurrence of x by $3y$.

In certain cases, we might have some irregular restriction on variables. For example, x should be exactly one of 3, 5, 12, 17, 19: no other values are admissible. This can be formulated by adding binary variables corresponding to the possible values of x. Let $\delta_3, \delta_5, \delta_{12}, \delta_{17}, \delta_{19}$ denote the indicator variables, e.g. $\delta_5 = 1$ if $x = 5$ and 0 otherwise. We need to enforce that exactly one of these is true:

$$\delta_3 + \delta_5 + \delta_{12} + \delta_{17} + \delta_{19} = 1.$$

Then we can define x the following way:

$$x = 3\delta_3 + 5\delta_5 + 12\delta_{12} + 17\delta_{17} + 19\delta_{19}.$$

This will guarantee that x takes exactly one of the prescribed values. These prescribed values do not necessarily need to be integers but they can also be real numbers.