Fundamentals of Operational Research

University of Edinburgh

Year 2022-2023

Lecturer: Sergio García Quiles

5th December 2022

Contents

Course Details

- Lectures: Mondays, 14:10-16:00. Room: JCMB Lecture Theatre A.
- Tutorials: Tuesdays, 14:10-15:00 and 15:10-16:00. Rooms: JCMB 3212 and 3217.
- Sometimes schedules may be changed at short notice. Please, check your email daily (Monday to Friday).
- Assessment: 80% exam, 20% continuous assessment.
- Continuous assessment: There are two assignments on Weeks 4 and 8 to be handed in one week later. Each is worth 10% of the final mark.
- We will follow my lecture notes and the course is self-contained. However, the following book provides additional support for those who want to know more:
  - *Introduction to Operations Research, 11th edition.* F.S. Hillier and G.J. Lieberman. McGraw Hill (2021).

Chapter 1

Introduction

This course consists of three parts, each of which could become a full course by itself: dynamic programming, integer programming, and game theory. The goal of this course is to introduce some basics notions of each of these subjects. Note that each of these three areas is sufficiently broad and important so that a single course devoted to the topic would not be enough to learn all.

The name dynamic programming was coined by Robert Bellman in 1957. It is an area that studies problems where sequential decisions have to be taken, for example, at different stages over time. It can be seen as recursive optimization, as we usually have an algorithm for solving the problem that is based on a recursive formula.

Integer programming studies problems where some (or all) of the decision variables are restricted to take integer values. The discrete nature of these problems makes it necessary to develop special techniques to solve them, as applying continuous linear programming usually leads to solutions that are not integer.

Finally, game theory studies strategic decisions where there are interactions among agents that are assumed to be rational.

Chapter 2

Dynamic Programming

The name dynamic programming was coined by Robert Bellman in 1957. It is an area that studies problems where sequential decisions have to be taken, for example, at different stages over time. It can be seen as recursive optimization. There is not such a thing as a general formulation for dynamic programming problems. It is a general type of methodology and the particular "formula" that must be employed at each problem must be developed when the main elements are identified.

Dynamic Programming is a useful technique to solve optimization problems which can be modelled in a recursive way. In order to illustrate this, let us start with an example.

Example 2.1 (The Hiker Problem)
A hiker would like to travel on foot from Edinburgh to Ben Nevis in a 4-day excursion. He would like to follow the least dangerous route. In order to do that, he has gathered the information on the different legs of this trip and has given a number to each of them. The lower the number, the lower the level of danger of that leg. The total level of danger of a route is equal to the sum of the levels of danger of the legs that constitute that route.

Each of the potential stops (nodes) for this excursion has been labelled with a letter. Levels of danger are given to the edges joining pairs for valid legs of the trip. See Figure 2.1.

The information is also shown in the following matrices:

$$
\begin{array}{c|c|c|c|}
 & B & C & D \\ \hline
A & 2 & 4 & 3 \\ \hline
\end{array}
\qquad
\begin{array}{c|c|c|c|}
 & E & F & G \\ \hline
B & 7 & 4 & 6 \\ \hline
C & 3 & 2 & 4 \\ \hline
D & 4 & 1 & 5 \\ \hline
\end{array}
\qquad
\begin{array}{c|c|c|}
 & H & I \\ \hline
E & 1 & 4 \\ \hline
F & 6 & 3 \\ \hline
G & 3 & 3 \\ \hline
\end{array}
\qquad
\begin{array}{c|c|}
 & J \\ \hline
H & 3 \\ \hline
I & 4 \\ \hline
\end{array}
$$

What route should the hiker follow to minimize the total level danger of that route?
Solution:
First, note that we can build a route where for each leg of the trip we go to the destination with the least danger from our current location. This is what we call in optimization a greedy heuristic algorithm.

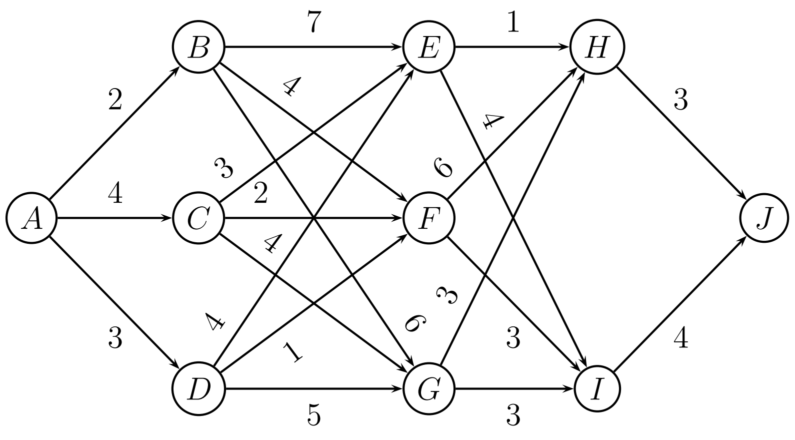


Figure 2.1: Hiking routes

This route is $A \to B \to F \to I \to J$ and its total level of danger is $2 + 4 + 3 + 4 = 13$.

However, we will see later that this route is not optimal, that is, this route does not offer the minimum total danger.

Now, in order to solve the problem we are going to solve several smaller subproblems. First, we will discuss informally the idea behind the solution method that we will develop later in a more formal mathematical way.

*Assume that the hiker has only one more stage to go (*note that we are already using a keyword in dynamic programming: stage*). He is on either location H or on location I. In both cases, it is clear where to go next: location J, as this is the only possibility. There is really nothing to think about here.*

Next we enlarge the subproblem by adding the previous stage. Assume that we are on location E. Being on location F or G would be dealt with in the same way.

From E we can go to either H or I. And once we have finished this stage, that is, when we have moved to either H or I, then we are on the previous case and we know what to do.

Thus, the optimal path from E to J is the one with the minimum total danger among the following options:

- *Leg $E \to H$ and then the optimal path from H to J.*
- *Leg $E \to I$ and then the optimal path from I to J.*

In this case, it is easy to calculate that the optimal path from E to J is $E \to H \to J$.

In the same way we can find the optimal paths from F to J and from G to J.

This idea can be extended to find optimal paths from B, C, and D to J with the information already calculated. Finally, an optimal path from A can be computed.

Now that we understand the main idea to develop the algorithm to solve the problem, we will write it down explicitly. But first we need to introduce some notation.

Key Property.

The key property that we will use is the following: Assume that we have found the path of least danger between j and k. Assume that we add now arc (i, j) at the beginning of this path. Then this new path is the path of least danger from i to k for which (i, j) is its first arc.

In order to see this, we denote by $D_j(i,k)$ the danger of a path from i to k for which its first arc is (i,j). The length of a path that minimizes that value is denote by $D_j^*(i,k)$. Also, $D(j,k)$ and $D^*(j,k)$ are the lengths of those paths when arc (i,j) is dropped. In addition, d_{ij} is the danger of arc (i,j).

Then:

$$D_j(i,k) = d_{ij} + D(j,k) \geq d_{ij} + D^*(j,k) = D_j^*(i,k).$$

The equalities above are a consequence that the length of a path is the sum of its two partial paths. The inequality is because, by definition, $D^*(j,k)$ minimizes $D(j,k)$.

Formulation.
Let x_n, $n = 1,2,3,4$, be the destination at the end of stage n.

In our case, we are looking for a route $A \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$, where we already know that $x_4 = J$.

Let $f_n(s,x_n)$ be the total danger of the best path for the remaining stages when the hiker is on location s about to start stage n and chooses x_n as his next destination. That is, the next leg of the trip will be $s \rightarrow x_n$.

For given s and x_n, let x_n^* be a value that minimizes $f_n(s,x_n)$. Note that x_n^* may not be unique. *Let $f_n^*(s)$ be that minimum danger. That is,*

$$f_n^*(s) = f_n(s,x_n^*) = \min\left\{f_n(s,x_n)\right\}_{x_n},$$

where

$$\begin{aligned} f_n(s,x_n) &= \textit{immediate danger (stage } n\textit{)} + \textit{minimum future danger (stages } n+1 \textit{ onwards)} \\ &= d_{sx_n} + f_{n+1}^*(x_n). \end{aligned}$$

The goal of the hiker is to calculate $f_1^*(A)$ and to find a route associated to that value. Note that there may be more than one optimal route for the optimal value $f_1^*(A)$.

*What dynamic programming does is to find successively $f_4^*(s)$, $f_3^*(s)$, and $f_2^*(s)$ for each of the possible states s (*note that we have introduced another dynamic programming term: state, when we refer to location in this example*). Finally, $f_2^*(s)$ values are used to obtain $f_1^*(A)$.*

Solution Algorithm.
We begin when the hiker is ready to start the last stage, $n = 4$. He has only one more stage to go. The route for the last stage is completely determined by his current state, $s = H$ or $s = I$, and his final destination, $x_4^* = J$.

As the route when starting on state s is $s \rightarrow J$ and $f_4^*(s) = f_4(s,J) = d_{sJ}$, the solution for stage 4 is given in Table 2.1.

s	$f_4^*(s)$	x_4^*
H	3	J
I	4	J

Table 2.1: $n = 4$.

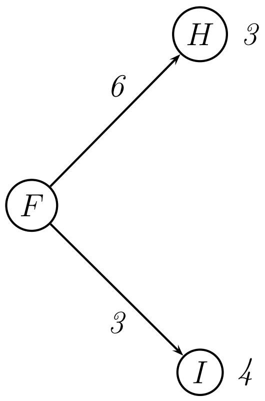
That was a very easy step of the algorithm. Let us see what happens when the hiker is about to start stage $n = 3$. Some additional calculations are required.

Assume, for example, that the hiker is in state F. Then he must go either to state H or to state I. The immediate levels of danger are $d_{FH} = 6$ *and* $d_{FI} = 3$*, respectively.*

If he chooses to go to state H, the minimum additional level of danger after he reaches that state is $f_4^*(H) = 3$*, which we have calculated earlier. As a consequence, the total level of danger of taking this route is* $6 + 3 = 9$*.*

If instead he chooses state I, the level of danger is $3 + 4 = 7$*.*

Therefore, the optimal choice is $x_3^* = I$ *and the minimum cost is* $f_3^*(F) = 7$*.*



We do analogous calculations for the other two states, E and G, when there are two stages to go. With that, we obtain Table 2.2 for stage 3:

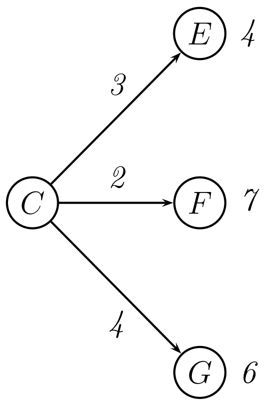
$s \backslash x_3$	$f_3(s,x_3) = d_{sx_3} + f_4^*(x_3)$		$f_3^*(s)$	x_3^*
	H	I		
E	$\underline{4}$	8	4	H
F	9	$\underline{7}$	7	I
G	$\underline{6}$	7	6	H

Table 2.2: $n = 3$.

Note that in the table we have underlined those decisions that are optimal among the subset of options that we are analyzing. For example, for state E, 4 is better than 8.

Now that we have solved the third stage $n = 3$*, we solve the second stage* $n = 2$ *in exactly the same way. For this stage, we have that* $f_2(s,x_2) = d_{sx_2} + f_3^*(x_2)$*. Precisely, we have just calculated the values* $f_3^*(x_2)$ *that will be used in this formula. This is why we are proceeding backwards in our algorithm.*

For example, let us assume that we are in state C:



The options from C are to go to either E, F, or G. The immediate levels of danger are $d_{CE} = 3$, $d_{CF} = 2$, and $d_{CG} = 4$, respectively. After having changed to those states, we have already calculated that the minimum additional levels of danger are $f_3^*(E) = 4$, $f_3^*(F) = 7$, and $f_3^*(G) = 6$, respectively. See Table 2.2.

Therefore, for the following decisions we can compute the minimum levels of danger associated to the paths that start on those states:

- $x_2 = E,\ f_2(C, E) = d_{CE} + f_3^*(E) = 3 + 4 = 7.$

- $x_2 = F,\ f_2(C, F) = d_{CF} + f_3^*(F) = 2 + 7 = 9.$

- $x_2 = G,\ f_2(C, G) = d_{CG} + f_3^*(G) = 4 + 6 = 10.$

We can see that the minimum of these three values is 7. This means that the minimum level of danger from state C to state J, the final destination, is $f_2^*(C) = f_2(C, E) = 7$, and that the immediate destination from C should be $x_2^* = E$.

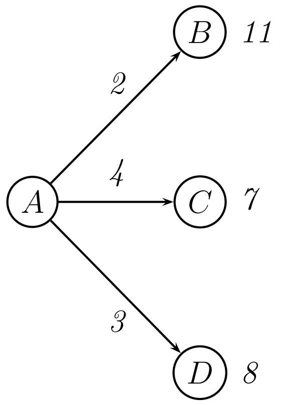
We do similar calculations for starting on states B or D and we gather all the information on Table 2.3.

x_2 / s	$f_2(s, x_2) = d_{sx_2} + f_3^*(x_2)$			$f_2^*(s)$	x_2^*
	E	F	G		
B	$\underline{11}$	$\underline{11}$	12	11	E or F
C	$\underline{7}$	9	10	7	E
D	$\underline{8}$	$\underline{8}$	11	8	E or F

Table 2.3: $n = 2$.

From Table 2.3, we can see that E and F tie as the minimizing decisions for x_2. Thus, from B we should go to either E or F. The same happens when starting on D.

Finally, we move on to the first stage of the problem: $n = 1$. We have four stages to go. The calculations are somehow reduced because now we have only one possible starting state:



As on previous occasions, we compute the total level of danger for the different alternatives:

- $x_1 = B,\ f_1(A, B) = d_{AB} + f_2^*(B) = 2 + 11 = 13.$
- $x_1 = C,\ f_1(A, C) = d_{AC} + f_2^*(C) = 4 + 7 = 11.$
- $x_1 = D,\ f_1(A, D) = d_{AD} + f_2^*(D) = 3 + 8 = 11.$

We can write this information as Table 2.4

$s \backslash x_1$	$f_1(s, x_1) = d_{sx_1} + f_2^*(x_1)$			$f_1^*(s)$	x_1^*
	B	C	D		
A	13	$\underline{11}$	$\underline{11}$	11	C or D

Table 2.4: $n = 1$.

Finally, by using the tables we can identify an optimal solution to our problem with total level of danger equal to 11:

- *For $n = 1$, we start on location A and our choices are either C or D.*
- *Let assume that we choose $x_1^* = C$.*
- *Then, for $n = 2$ we have that $x_2^* = E$.*
- *Next, for $n = 3$ we have that $x_3^* = H$.*
- *Last, for $n = 4$ we have that $x_4^* = J$.*

The route is $A \to C \to E \to H \to J$ Note that if we choose $x_1^* = D$, then we have two more optimal routes: $A \to D \to E \to H \to J$ and $A \to D \to F \to I \to J$. All of them have the same optimal value: 11.

2.1. Concepts of Dynamic Programming

Now that we are familiar with the kind of problems that dynamic programming can solve, let us introduce some basic concepts more formally.

In dynamic programming, we divide a problem in **stages**. Each stage needs a **policy decision**. In the hiker example, we had four stages, one for each leg of the trip. The policy decision for each stage was to decide to which location the hiker should go next.

In each stage there is a number of **states** associated with the beginning of the stage. The states are the possible conditions in which the system may be when starting each stage. In the hiker example, the states at each stage are the possible locations from where we can start on that stage.

The policy decision seeks to modify the state of the current stage to a state associated to the start of the next stage. For some applications like the example that we have just seen, if we see our problem represented with a network, at each stage we are deciding which edge (or arc if the graph is directed) to use. Each node of the network is a state. And, if we sort in a certain way the nodes of this network, we can have that the nodes of each column are the states of a stage. For many applications, we are looking at finding the shortest or the longest path through this network. But note that, this that was valid for the hiker example, may not be possible for other problems.

Our ultimate goal is to find an **optimal policy** for the overlall problem. This is a collection of optimal decisions for each stage for each of the possible states so that a certain objective function is maximized or minimized. In our hiker problem, we have constructed tables for the different stages. In each table we calculated for each state the best decision. The advantage of this is that, if something happens and the hiker needs to deviate from the optimal route that is following, then he will know how to complete optimally the rest of the deviated route. All this was done to find a route for which the total accumulated danger is minimum (our objective function).

The most important aspect is that, given the current state, an optimal policy for the remaining stages is independent of the policy decisions adopted in previous stages. This means that the optimal immediate decision depends only on the current state and not on how we arrived in that state. This is known as the **principle of optimality** for Dynamic Programming. We can see this in our hiker problem: once at a given location, the total level of danger of the remaining route does not depend on the legs travelled to arrived to our current location. This is exactly what we proved when we were discussing the "key property" of the problem.

The solution method finds first the optimal policy for the last state. This involves stating the optimal policy decision for each of the possible states of that stage. Usually, as in the case of our hiker problem, this is easy.

Then, a **recursive relationship** identifies the optimal policy for stage n given that the optimal policy for stage $n+1$ is known. In the case of the hiker problem, this was given by the formula

$$f_n^*(s) = \min \left\{ d_{sx_n} + f_{n+1}^*(x_n) \right\}_{x_n} .$$

The main difficulty if that the exact expression for the recursive relationship differs from one dynamic programming problem to another. However, the core ideas are the same and we will often use the core notation that we have already used in the hiker example.

- N = number of stages.
- n = label for the current stage, $n = 1, \ldots, N$.
- s_n = current state for stage n.
- x_n = decision variable for stage n.
- x_n^* = optimal value of x_n given s_n.
- $f_n(s_n, x_n)$ = contribution of stages $n, n+1, \ldots, N$ to the objective function if the system starts in state s_n at stage n, the immediate decision is x_n, and optimal decisions are made thereafter.
- $f_n^*(s_n) = f_n(s_n, x_n^*)$.
- The recursive relationship will be of the form

$$f_n^*(s_n) = \max \left\{ f_n(s_n, x_n) \right\}_{x_n} \quad \text{or} \quad f_n^*(s_n) = \min \left\{ f_n(s_n, x_n) \right\}_{x_n} ,$$

where $f_n(s_n, x_n)$ is written as a function of s_n, x_n, $f^*_{n+1}(s_{n+1})$, and probably some measure of the immediate contribution of x_n to the objective function.

When the current stage number n is decreased by 1, the new $f^*_n(s_n)$ function is obtained by using the $f^*_{n+1}(s_{n+1})$ that was computed on the previous iteration.

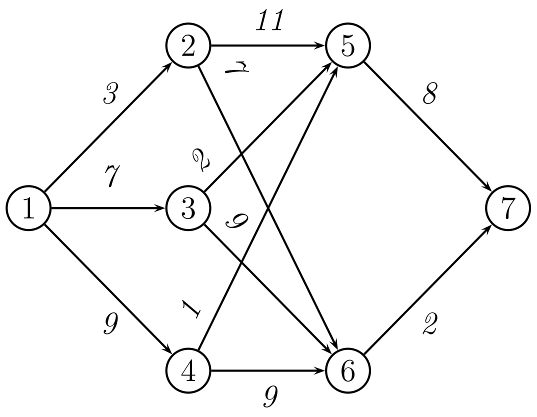
When Dynamic Programming Does Not Apply

Let us see now an example where dynamic programming cannot be used.

Example 2.2 (Not with Dynamic Programming)

Given a path on a network, we define the range of the path as the difference between the largest arc and the shortest arc. For example, in the hiker problem, if we now consider the value associated to each arc as its length, the range of path $A \to B \to E \to H \to J$ is $7 - 1 = 6$.

Let us find the path with minimum range on the following graph:



Let us use total enumeration to list the 6 paths and its ranges:

- $1 \to 2 \to 5 \to 7$. *The range is 8.*
- $1 \to 2 \to 6 \to 7$. *The range is 5.*
- $1 \to 3 \to 5 \to 7$. *The range is 6.*
- $1 \to 3 \to 6 \to 7$. *The range is 7.*
- $1 \to 4 \to 5 \to 7$. *The range is 8.*
- $1 \to 4 \to 6 \to 7$. *The range is 7.*

The optimal values is 5 *and the optimal path is* $1 \to 2 \to 6 \to 7$.

If we could apply dynamic programming, and, particularly, if the principle of optimality held, it would happen that the partial path between nodes 2 and 7 in this optimal path would be optimal for that restricted route. But we will see that this is not the case.

Let us enumerate now the paths from node 2 to node 7:

- $2 \to 5 \to 7$. *The range is 3.*
- $2 \to 6 \to 7$. *The range is 5.*

The optimal path is $2 \to 5 \to 7$*, but this is not the partial path between 1 and 7 from the global optimal path.*

This violates the principle of optimality, as it does not hold that the optimal bath between nodes 1 and 7 is an arc $(1, j)$ for some node j and then the optimal path from j to 7. Therefore, we cannot apply dynamic programming to solve this problem.

2.2. The Critical Path Method

The **Critical Path Method (CPM)** is an optimization technique used to analyze a flow of activities in a project. We would like to organize clearly the sequence of activities in a diagram (a network), decide how long it will take for the project to be completed, and study what is the earliest and latest time that the given activities can start and end. This network has a start node and and end node, and we would like to find the longest path from the start to the end.

We will learn this technique with an example.

Example 2.3 (The Construction Project)
Edinburgh Constructors S.L. is a constructor company whose most recent winning project is to build a new plant for a local manufacturer, who needs the plant in operation as soon as possible. The list of activities that need to be completed is given in Table 2.5.

Activity Label	Activity Description	Immediate Predecessors	Duration (in weeks)
A	Excavate	-	2
B	Lay the foundation	A	4
C	Put up the rough wall	B	10
D	Put up the roof	C	6
E	Install the exterior plumbing	C	4
F	Install the interior plumbing	E	5
G	Put up the exterior siding	D	7
H	Do the exterior painting	E,G	9
I	Do the electrical work	C	7
J	Put up the wallboard	F,I	8
K	Install the flooring	J	4
L	Do the interior painting	J	5
M	Install the exterior fixtures	H	2
N	Install the interior fixtures	K,L	6

Table 2.5: List of activities.

*For any given activity, its **immediate predecessors** are those activities that must be completed by no later than the starting time of the given activity. In a similar way, this activity is named an **immediate successor** for each of its immediate predecessors.*

For example, the first and second entries of the table tells us the following:

- *The excavation activity does not need to wait for any other activity.*
- *The excavation must be completed before the activity lay the foundation can begin.*
- *The excavation takes 2 weeks and laying the foundation takes 4 weeks.*

If an activity has more than one predecessor, all of them must be finished before the activity can begin. For example, before we can start with the exterior painting (activity H), we must have finished installing the exterior plumbing (activity E) and putting up the exterior siding (activity G).

Something that we can easily observe is that the sum of all the activity durations is 79 weeks, but it is likely that the project can be finished much earlier than that. This is the time necessary *if the activities are done sequentially, that is, only one activity is done at any given time and no activity begins before that activity has been finished. The key to meet the deadline is that some of the activities can be done in parallel. With the CPM analysis we will be able to determine what order the activities must follow and which activities can be done in parallel.*

Now we are interested in two things: how to represent the activities of the project so that it is visually easy to understand the sequence of the activities, and how to calculate the time required to complete the project.

Project Network

A project network is a network used to represent a work project. In the network we have nodes and arcs whose meaning depends on the way used to represent the activities of the project (we will elaborate this later). The information that we need is:

- *Activity details: the project needs to be broken down into individual activities.*
- *Precedence relationships: identify the immediate predecessor(s) for each activity.*
- *Activity duration: estimated time that will take to complete each activity.*

Network Representation

We have two different project networks depending on how we represent the activities:

- *Activity-on-arc (AOA) project network. Each activity is represented by an arc. Nodes are used to separate activities (outgoing arcs) from their immediate predecessors (incoming arcs).*
- *Activity-on-node (AON) project network. Each activity is represented by a node. The arcs show the precedence relationships. Particularly, each activity node has an arc coming from each of its immediate predecessors.*

AON project networks are more popular than AOA networks. They are easier to construct, understand, and revise. In this course we will study only AON project networks.

Usually we are given the list of activities as a table like Table 2.5. We then construct the AON project networks as follows:

1. *Draw one node for "Start" and one node for "Finish". Draw one node for each activity from the list of activities.*
2. *Draw arcs to each node from its immediate predecessors.*
3. *For each node with no incoming arcs draw an arc to it from the "Start" node.*
4. *For each node with no outgoing arcs draw an arc from it to the "Finish" node.*

We can see the AON project network representation for our example in Figure 2.2.

Note that only the nodes have values (the duration of the activity represented by that node). Arcs just show the precedence relationships.

The Critical Path

The key questions that we would like to answer are:

1. *What is the total time required to complete the project if no delays occur?*
2. *When do the individual activities need to start and finish (at the latest) to meet this project completion time?*
3. *When can the individual activities start and finish (at the earliest) if no delays occur?*
4. *Which are the critical bottleneck activities where any delays must be avoided to prevent delaying project completion?*

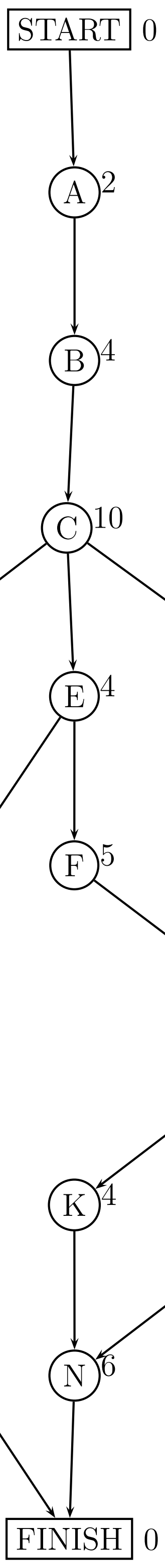


Figure 2.2: AON project network representation.

5. *For the other activities, how much delay can be tolerated without delaying project completion?*

In the case of an AON project network, the length of a path is the sum of the durations of the activities that are in the path.

In order to find out how long it will take to complete the project, we calculate the length of the longest path from the "Start" node to the "Finish" node. This path is named ***critical path*** *(or paths, if there is more than one) because all its activities are critical: a delay in any of those activities will delay the project. Knowing this is very useful to a project manager to focus his attention on those activities to avoid delays. Questions 1 and 4 above are answered by finding the critical path.*

In order to use dynamic programming to find the longest path from the "Start" node to the "Finish" node, we identify each stage with a node. There is only one state per stage. Then we can use either forward or backward dynamic programming to compute this longest path.

For example, let us use first backward dynamic programming.

For stage i, f_i^* will denote the length of the longest path from node i to node "Finish".

We have that $f_{Finish}^* = 0$ and we have the recursive expression

$$f_i^* = d_i + \max\{f_j^*\}_{(i,j)\in A},$$

where d_i is the duration of activity i and A is the set of arcs of the AON project network. We have the calculations on Table 2.6.

i	f_i^*	Next node is
Finish	0	-
N	$6+0=6$	Finish
M	$2+0=2$	Finish
L	$5+6=11$	N
K	$4+6=10$	N
J	$8+\max\{10,11\}=19$	L
I	$7+19=26$	J
H	$9+2=11$	M
G	$7+11=18$	H
F	$5+19=24$	J
E	$4+\max\{11,24\}=28$	F
D	$6+18=24$	G
C	$10+\max\{24,28,26\}=38$	E
B	$4+38=42$	C
A	$2+42=44$	B
Start	$0+44=44$	A

Table 2.6: Critical path with backward dynamic programming.

We have that the length of the longest path is 44. Therefore, it will take 44 weeks to complete the project. The critical path is

$$\textit{Start}\rightarrow A\rightarrow B\rightarrow C\rightarrow E\rightarrow F\rightarrow J\rightarrow L\rightarrow N\rightarrow \textit{Finish}.$$

We can also find the critical path with forward dynamic programming. In this case, for stage i we use f_i^* to represent the length of the longest path from node "Start" to node i.

We have that $f^*_{Start} = 0$ and we have the recursive expression

$$f^*_j = d_j + \max\{f^*_i\}_{(i,j)\in A}.$$

We have the calculations on Table 2.7.

i	f^*_i	Previous node is
Start	0	-
A	$2+0=2$	Start
B	$4+2=6$	A
C	$10+6=16$	B
D	$6+16=22$	C
E	$4+16=20$	C
F	$5+20=25$	E
G	$7+22=29$	D
H	$9+\max\{20,29\}=38$	G
I	$7+16=23$	C
J	$8+\max\{25,23\}=33$	F
K	$4+33=37$	J
L	$5+33=38$	J
M	$2+38=40$	H
N	$6+\max\{37,38\}=44$	L
Finish	$0+\max\{40,44\}=44$	N

Table 2.7: Critical path with forward dynamic programming.

Obviously, we obtain the same critical path and the same length as before.

Scheduling Individual Activities

Those activities on the critical path have to be started and finished with no room for changes. But what about those activities that are not on the critical path? We may be more flexible for those activities.

The start and finish times if no delays occur anywhere on the project are name the earliest start time (ES) and the earliest finish time (EF) of the activity.

It is immediate that

$$EF = ES + \textit{duration of the activity}.$$

We start our time horizon at $t = 0$, which is the start time of the project.

On our AON project network, the earliest finish time of activity j is the length of the longest path from the "Start" node to node j:

$$EF(j) = LP(Start, j),$$

where $LP(a, b)$ is the length of the longest path from node a to node b.

The earliest start time of an activity is equal to the largest of the earliest finish times of its immediate predecessors:

$$ES(j) = \max\{EF(i)\}_{(i,j)\in A}.$$

Or, as we defined above, we can also write

$$EF(j) = ES(j) + d_j.$$

We can use the information that we calculated when we computed the critical path with forward dynamic programming in Table 2.7 to obtain Table 2.8.

i	$EF(i)$	$ES(i)$
Start	0	0
A	2	0
B	6	2
C	16	6
D	22	16
E	20	16
F	25	20
G	29	22
H	38	29
I	23	16
J	33	25
K	37	33
L	38	33
M	40	38
N	44	38
Finish	44	44

Table 2.8: Earliest start and finish times.

We are also interested in knowing what happens if some activity takes longer than expected. Will this delay the project? This depends on the activity and the length of the delay. thus, let us analyze how much later can activities start or finish without delaying the project.

The latest start time (LS) for an activity is the latest time that it can start without delaying the completion of the project, that is, the "Finish" node is still reached at its earliest start time. When we perform this calculation, we assume no other delays in the project. Analogously, we define the latest finish time (LF) with respect to finishing the activity.

It is immediate that

$$LF = LS + \textit{duration of the activity.}$$

In terms of longest paths, the latest start time for activity i is equal to the length of the critical path minus the length of the longest path from node i to node "Finish".

We can sue the information that we calculated when we computed the critical path with backward dynamic programming in Table 2.6 to obtain Table 2.9.

Finally, we measure how much flexibility we have with an activity by calculating its slack (S). This is calculated as the difference between its latests finish time and its earliest finish time:

$$S(i) = LF(i) - EF(i).$$

i	$LS(i)$	$LF(i)$
Start	$44 - 44 = 0$	0
A	$44 - 44 = 0$	2
B	$44 - 42 = 2$	6
C	$44 - 38 = 6$	16
D	$44 - 24 = 20$	26
E	$44 - 28 = 16$	20
F	$44 - 24 = 20$	25
G	$44 - 18 = 26$	33
H	$44 - 11 = 33$	42
I	$44 - 26 = 18$	25
J	$44 - 19 = 25$	33
K	$44 - 10 = 34$	38
L	$44 - 11 = 33$	38
M	$44 - 2 = 42$	44
N	$44 - 6 = 38$	44
Finish	$44 - 0 = 44$	44

Table 2.9: Latest start and finish times.

This value indicates how much beyond its earliest start time can an activity be delayed without delaying the completion of the project.

We have calculated in Table 2.10 the slacks for the different activities.

i	$EF(i)$	$LF(i)$	S(i)
Start	0	0	0
A	2	2	0
B	6	6	0
C	16	16	0
D	22	26	4
E	20	20	0
F	25	25	0
G	29	33	4
H	38	42	4
I	23	25	2
J	33	33	0
K	37	38	1
L	38	38	0
M	40	44	4
N	44	44	0
Finish	44	44	0

Table 2.10: Slacks.

Note that the activities on the critical path have slack equal to 0.

2.3. Inventory Problems

In an inventory problem we have some amount of one or more products. We need to decide how much of each product we take out or we add to our inventory during a time horizon. These problems can be solved in a natural way with dynamic programming because stages and states appear as an inherent part of the problem.

There are many different inventory problems. Some features that an inventory problem may have are:

- The amount of product: discrete (e.g., number of cars) or continuous (e.g., amount of gas).
- Number of different types of products (e.g., one car model or several models).
- Storage capacity and cost.
- Cost of adding more stock.
- Demand selling price of the products at each stage.

Let us see an example.

Example 2.4 (An Inventory of Cakes)
Edinburgh Bakery has a stock of 5 cakes, which are perishable goods. They need to be sold before 3 days or they will be wasted. By varying the price of the cakes and performing some targeted marketing, Edinburgh Bakery can control how many cakes will be sold on each day and at what profit. Table 2.11 shows the profit per number of cakes and day in which they are sold.

	Day		
	1	2	3
Discount factor:	1.0	0.7	0.4
Cakes sold	Profit		
0	0	0	0
1	100	70	40
2	170	119	68
3	220	154	88
4	240	168	96
5	240	168	96

Table 2.11: Profit per cakes and days.

What should the strategy be to maximize the total profit?

<u>Solution:</u>
We begin by observing that each state can be described by a pair (t, v), where t is the time period (day) and v is the number of items in stock when period t starts.

All the possible states can be organized as the graph shown in Figure 2.3. Each column corresponds to a stage and each node is a state. The arcs show the possible transitions. The label associated to each arc is the profit raised from that sale. Therefore, our original problem of finding the most profitable strategy is equivalent to finding the longest path in this directed network. Alternatively, we can use a recursive expression, which is what we will do here. Both approaches are equivalent.

Let $p(t, u)$ be the profit shown in Table 2.11 from selling u cakes on day t. Let $F(t, v)$ be the maximum profit at the start of day t when the amount of cakes in stock is v.

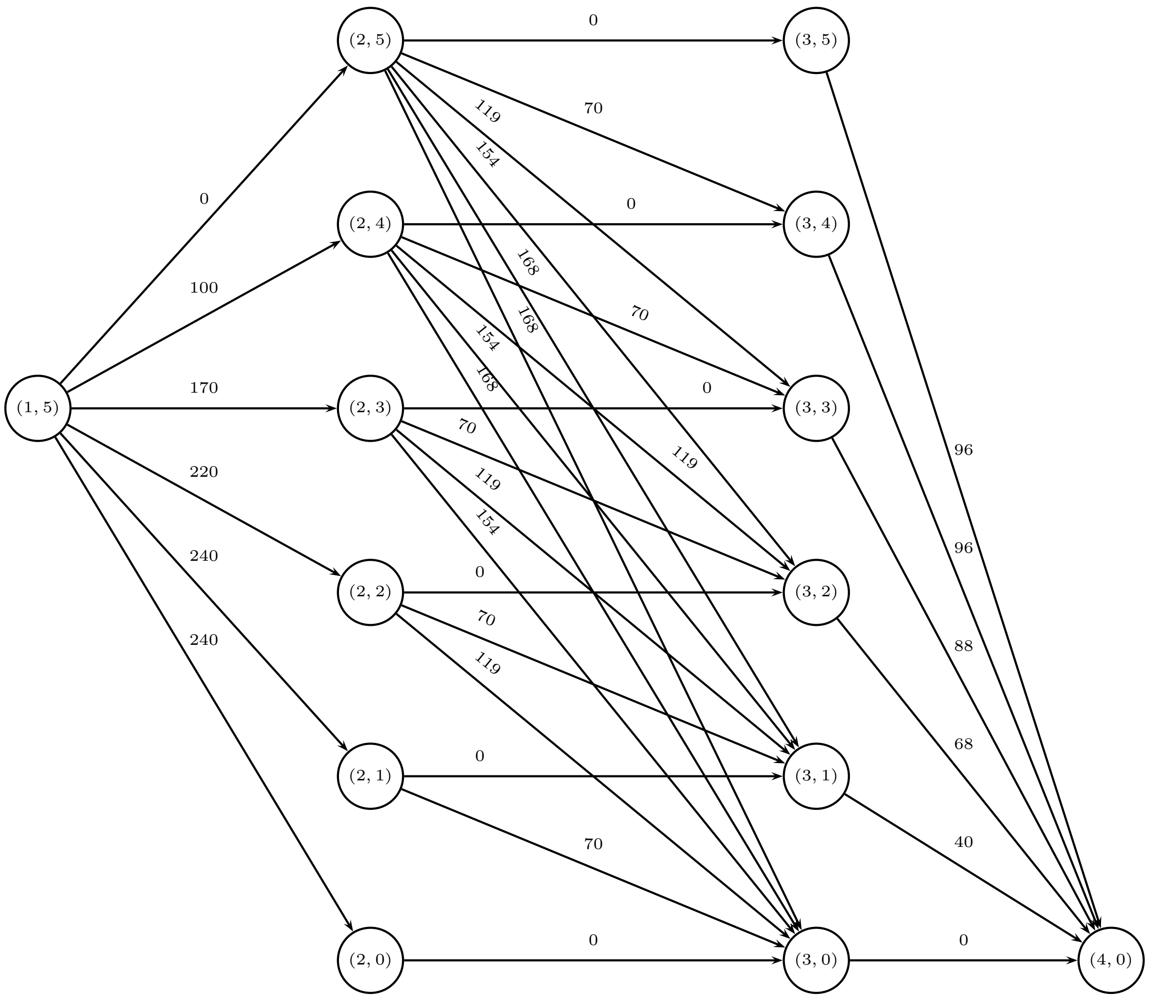


Figure 2.3: Network of states and transitions.

We can easily determine the maximum profit at each stage and for each state with the following recursive formula:

$$F(t,v) = \max\left\{p(t,u) + F(t+1, v-u)\right\}_{0 \le u \le v},$$

with $F(4,0) = 0$. Note:

- *How the term inside the maximum adds the profit from the current decision to the maximum profit that can be obtained from the following stage onwards while deducting from the future stock the u cakes that are sold on the current stage.*
- *$F(4,0) = 0$ because there is no profit in keeping stock at the end of the 3 days. It is always more profitable to sell all the cakes because no matter the amount or the day, the profits in Table 2.11 are always positive.*

Now we need to do some calculations based on the recursive expression that we have written. We will compute the maximum profit for every stage and every state.

1. *Stage $t = 4$.*
   There is nothing to do here. We know that $F(4,0) = 0$.
2. *Stage $t = 3$.*
   We have 6 possible states. We will have to do calculations for all of them. Luckily, they

are straightforward because for each state there is only one possibility: to sell all the cakes, as it has been discussed above.

- $F(3,0) = p(3,0) = 0$.
- $F(3,1) = p(3,1) = 40$.
- $F(3,2) = p(3,2) = 68$.
- $F(3,3) = p(3,3) = 88$.
- $F(3,4) = p(3,4) = 96$.
- $F(3,5) = p(3,5) = 96$.

3. *Stage $t = 2$.*
We have 6 possible states and for each of them there are several possibilities, from selling all the cakes to not selling any. The possible decisions are shown in Figure 2.3.
   - $F(2,0) = p(2,0) + F(3,0) = 0 + 0 = 0$.
   - $F(2,1) = \max\{p(2,1)+F(3,0), p(2,0)+F(3,1)\} = \max\{70+0, 0+40\} = \max\{\underline{70}, 40\} = 40$. *It is associated to selling 1 cake.*
   - $F(2,2) = \max\{p(2,2)+F(3,0), p(2,1)+F(3,1), p(2,0)+F(3,2)\} = \max\{119+0, 70+40, 0+68\} = \max\{\underline{119}, 110, 68\} = 119$. *It is associated to selling 2 cakes.*
   - $F(2,3) = \max\{p(2,3)+F(3,0), p(2,2)+F(3,1), p(2,1)+F(3,2), p(2,0)+F(3,3)\} = \max\{154+0, 119+40, 70+68, 0+88\} = \max\{154, \underline{159}, 138, 88\} = 159$. *It is associated to selling 2 cakes.*
   - $F(2,4) = \max\{p(2,4)+F(3,0), p(2,3)+F(3,1), p(2,2)+F(3,2), p(2,1)+F(3,3), p(2,0)+F(3,4)\} = \max\{168+0, 154+40, 119, 68, 70+88, 0+96\} = \max\{168, \underline{194}, 187, 158, 96\} = 194$. *It is associated to selling 3 cakes.*
   - $F(2,5) = \max\{p(2,5)+F(3,0), p(2,4)+F(3,1), p(2,3)+F(3,2), p(2,2)+F(3,3), p(2,1)+F(3,4), p(2,0)+F(3,5)\} = \max\{168+0, 168+40, 154+68, 119+88, 70+96, 0+96\} = \max\{168, 208, \underline{222}, 187, 166, 96\} = 222$. *It is associated to selling 3 cakes.*
4. *Stage $t = 1$.*
Finally, we do the calculations for the last stage of this backward algorithm. Now we have only one possible state because we know that our initial stock is 5 cakes.
$F(1,5) = \max\{p(1,5)+F(2,0), p(1,4)+F(2,1), p(1,3)+F(2,2), p(1,2)+F(2,3), p(1,1)+F(2,4), p(1,0)+F(2,5)\} = \max\{240+0, 240+70, 220+119, 170+159, 100+194, 0+222\} = \max\{240, 310, \underline{339}, 329, 294, 222\} = 339$. *It is associated to selling 3 cakes.*

Therefore, the maximum profit is 339. *This is obtained from doing as follows:*

1. *Sell 3 cakes on day 1.*
2. *Sell 2 cakes on day 2.*
3. *Do not sell any cake on day 3.*

As we have seen, the problem is not difficult to solve, but it involves a substantial number of calculations. This is one of the reasons why dynamic programming can be implemented quite successfully using your favourite programming language.

Now, let us tackle a variant of the previous problem where there are new traits that make the problem slightly more difficult (but not too much).

Example 2.5 (A Revisited Inventory of Cakes)
Solve the problem stated in Example 2.4 but with these two additional traits:

- *At most 3 cakes can be sold on a given day.*

- *At the start of any day that we still have some cakes left, we must pay an inventory cost of 50.*

<u>Solution:</u>
The fact that now there is a maximum on the number of cakes that can be sold per day makes that some decisions are not possible now (e.g., selling 5 cakes on the same day). In terms of the graph that he had drawn, that means that some arcs will not be present now. In terms of the recursive expression, it means that now we have a bound on the decision variable associated to how many cakes are sold on a given day. We will write the recursive expression later. First we work with the graph.

The modified graph can be seen in Figure 2.4. It is obtained by deleting the arcs that allow to sell 4 or 5 cakes on a single day.

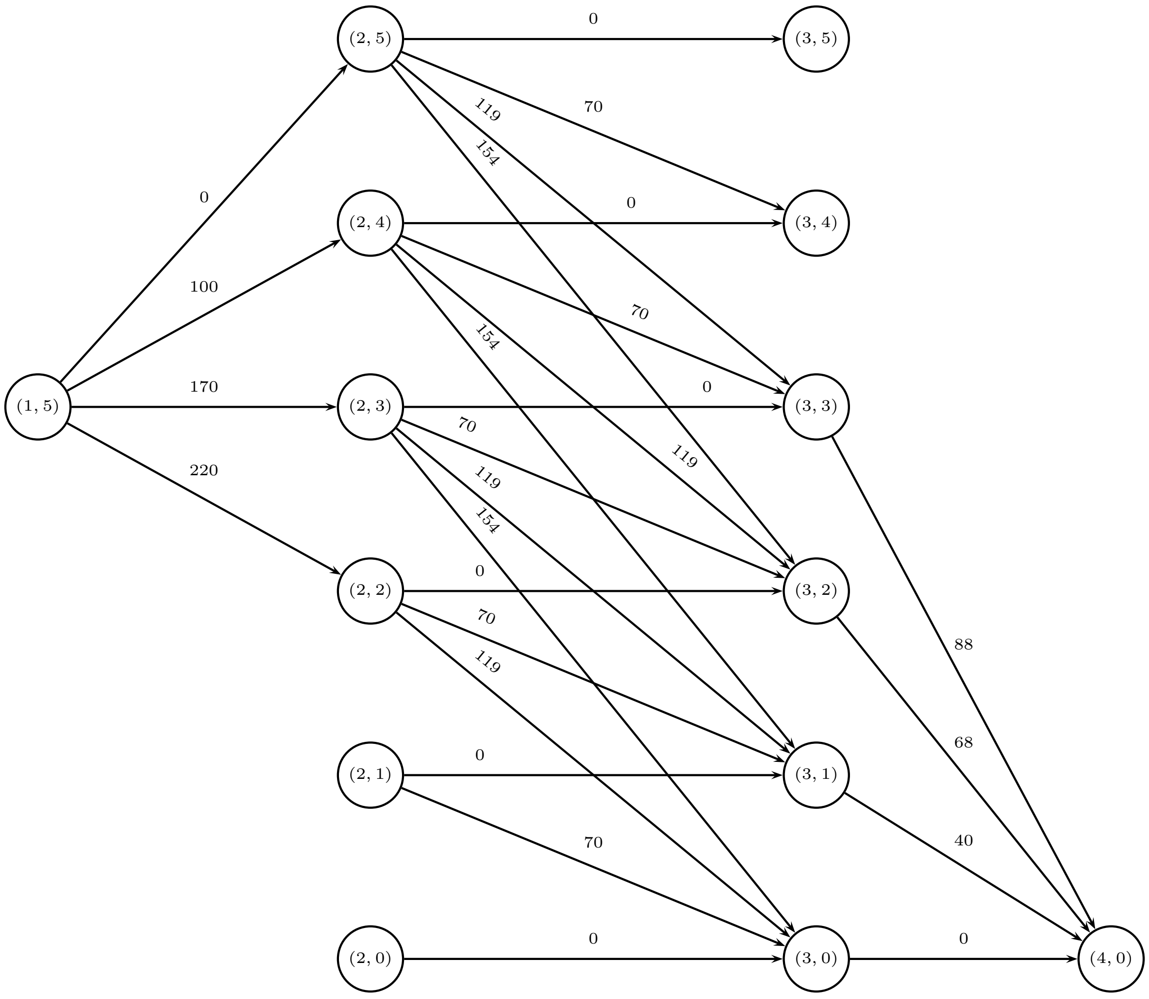


Figure 2.4: Network of states and transitions (some arcs removed).

However, we immediately observe that some nodes cannot be reached from the initial state, node $(1, 5)$, or cannot reach the final state, node $(4, 0)$. Those states make no sense now, since there is not point in having 5 cakes when we begin the third day or 1 cake when we start the second day. Thus, we can simplify the graph by removing these nodes and its incoming arcs. The result can be seen in Figure 2.5.

Figure 2.5 shows now the only possible transitions. But we have not included yet the inventory

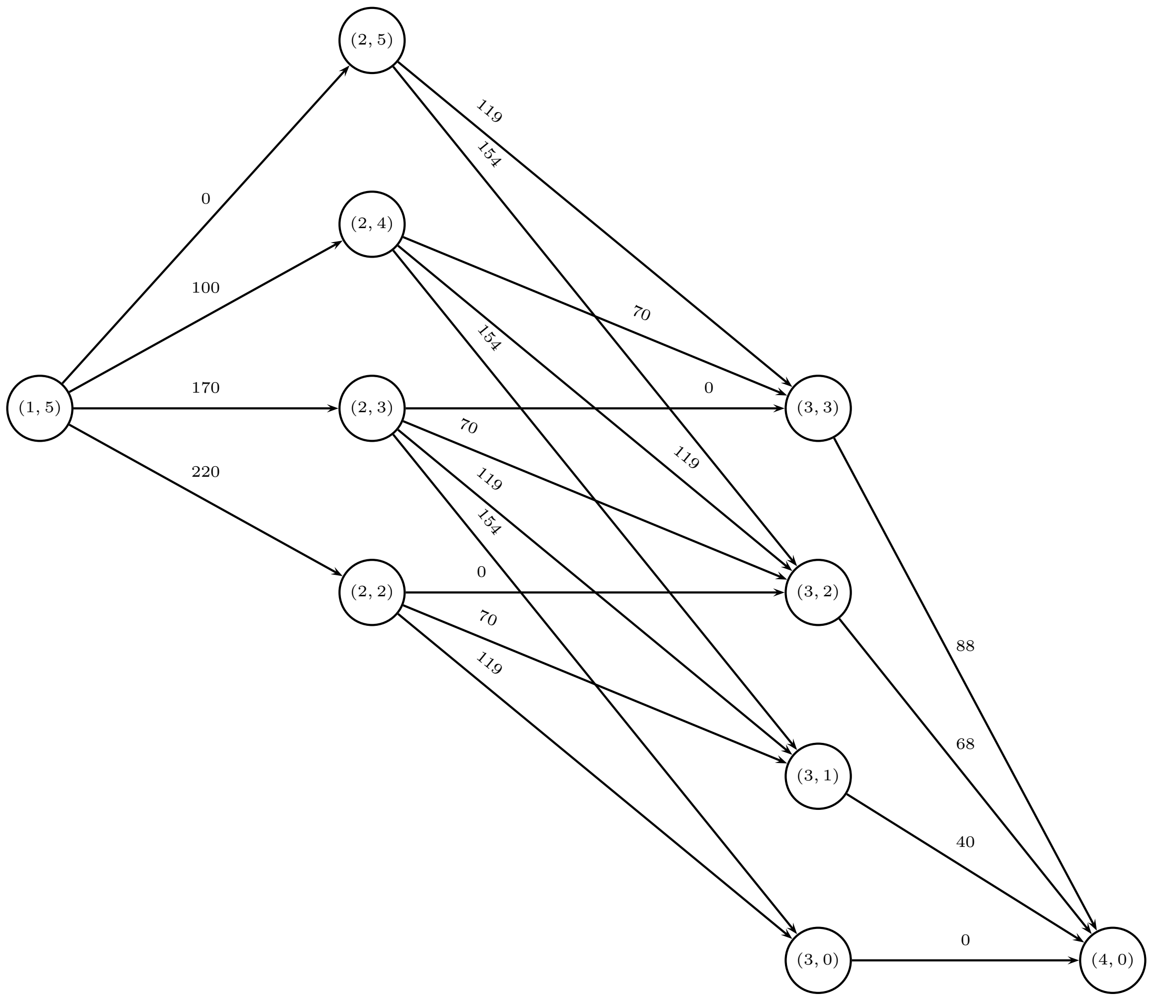


Figure 2.5: Network of states and transitions (some arcs and nodes removed).

cost. We can include this by deducting 50 from the profit of every arc that leaves from a state with strictly positive inventory. We can see the result in Figure 2.6. If we find a longest path in this graph, we will have solved the new problem.

Alternatively, we can work with a recursive formula. We can modify the expression that we used before to include the new traits of the problem. Let us do it in two steps.

First, we add that at most 3 cakes can be solved on a given day. Then, our recursive expression writes as follows:

$$F(t,v) = \max\left\{p(t,u) + F(t+1, v-u)\right\}_{0\le u\le v, u\le 3} = \max\left\{p(t,u) + F(t+1, v-u)\right\}_{0\le u\le \min\{v,3\}},$$

with $F(4,0) = 0$.

Now, we need to include that there is a cost of 50 units at any period that we start with some cakes in stock. Therefore, our recursive function will be:

$$F(t,v) = \max\left\{p(t,u) - \delta(v) + F(t+1, v-u)\right\}_{0\le u\le \min\{v,3\}},$$

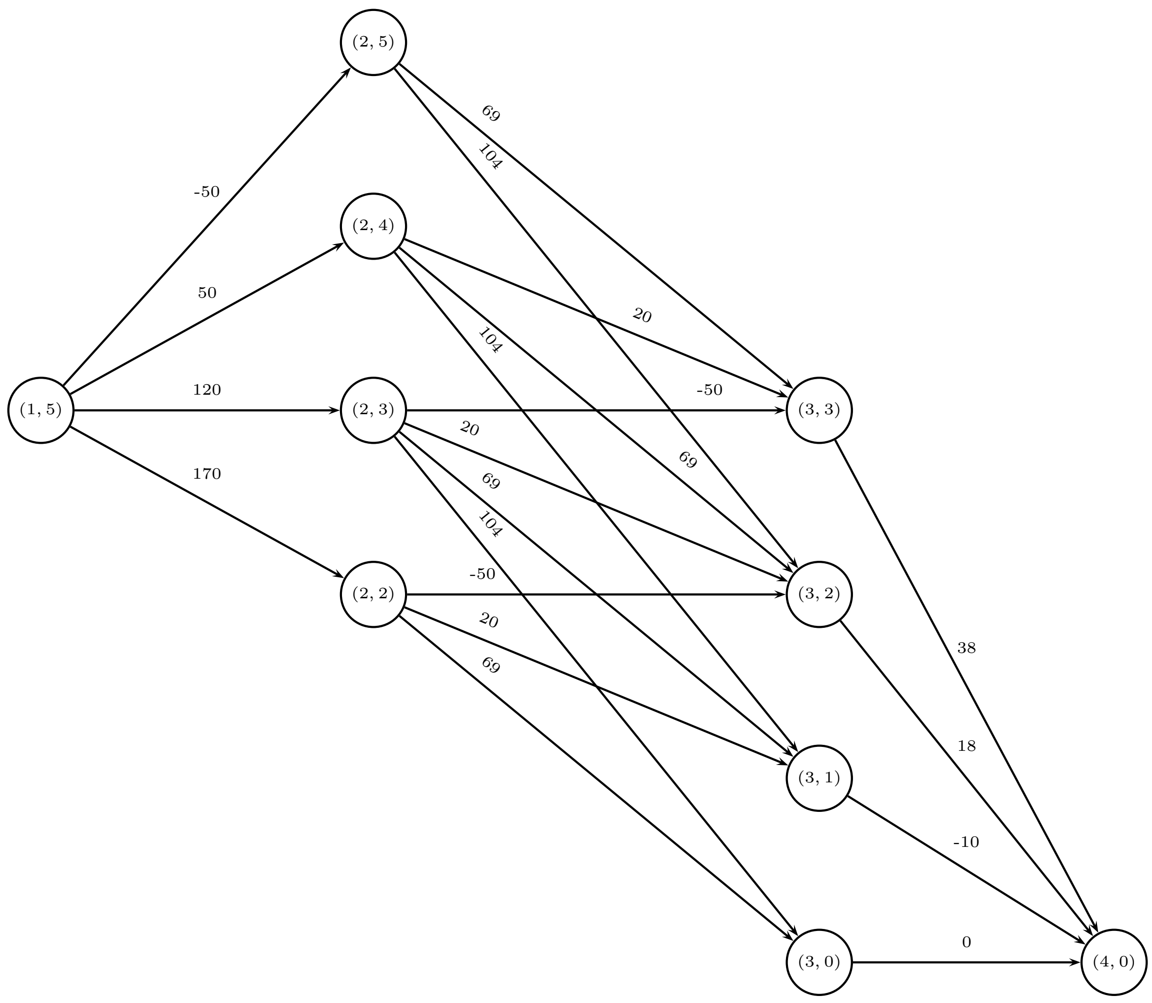


Figure 2.6: Network of states and transitions (labels updated).

with $F(4,0) = 0$ *and*

$$\delta(v) = \begin{cases} 0, & v = 0, \\ 50, & v \geq 1. \end{cases}$$

No matter how we solve the problem (longest path on Figure 2.6 or recursive formula), we find that the optimal decision is still the same (to sell 3 cakes on day 1 and 2 cakes on day 2). The new optimal cost is 239.

2.4. Allocation Problems

Sometimes we face the problem of allocating scarce resources to different activities that are required in order to achieve a certain objective. For example, what amount of raw materials to use in the production of chemical components to maximize the profit from selling the final products.

Although in these problems there are no stages in the sense of dividing a time horizon into days or hours, we can still use dynamic programming. Let us see an example.

Example 2.6 (Distributing Medical Teams to Countries)
The Lothian Health Council is devoted to improve health care in the underdeveloped countries. It has 5 medical teams that can be allocated to 3 countries to improve their global health care services. The performance is measured by the numbers given in Table 2.12. This is the additional person-years of life times the country's population in multiples of 1,000. The problem that the Lothian Health Council has to solve is how to allocate the teams so that the total performance is maximized. The number of teams allocated to any country must be an integer number, that is, a team cannot be split.

Number of	Country		
medical teams	1	2	3
0	0	0	0
1	45	20	50
2	70	45	70
3	90	75	80
4	105	110	100
5	120	150	130

Table 2.12: Thousands of Additional Person-Years of Life.

Solution:
We have three interrelated decisions: how many medical teams to allocate to each country. In order to apply dynamic programming, even if we do not have a fixed sequence, we can see each country as one stage. The decision variable x_n will represent how many medical teams should be allocated on stage n (how many teams will be allocated to country n).

For each stage the different states s_n are how many medical teams are still available. This means that:

- $s_1 = 5$.
- $s_2 = 5 - x_1$.
- $s_3 = 5 - x_1 - x_2$.

In addition, by looking at Table 2.12 we can see that there is no advantage in not allocating all the teams. Therefore, $x_1 + x_2 + x_3 = 5$ and the only relevant state at the end of the process (stage $n = 4$) is $s_4 = 0$. We can represent the different states and transitions with a directed graph. See Figure 2.7.

In order to solve this problem, we need to find a longest path from node $(1, 5)$ to node $(4, 0)$.

Alternatively, we can solve the problem by using a recursive expression. Let $p_i(x_i)$ be the benefit from allocating x_i medical teams to country i.

We would like to maximize $p_1(x_1) + p_2(x_2) + p_3(x_3)$ with the condition $x_1 + x_2 + x_3 = 5$ and all the x_i nonnegative integers.

Let $f_n(s_n, x_n)$ be the maximum yield from allocating x_n medical teams to country n and $x_{n+1} + \ldots + x_3$ to countries $n + 1, \ldots, 3$. Then:

$$f_n(s_n, x_n) = p_n(x_n) + \max\left\{ \sum_{i=n+1}^{3} p_i(x_i) \ / \ x_n + \ldots + x_3 = s_n,\ x_n, \ldots, x_3 \in \mathbb{Z}^+ \right\}.$$

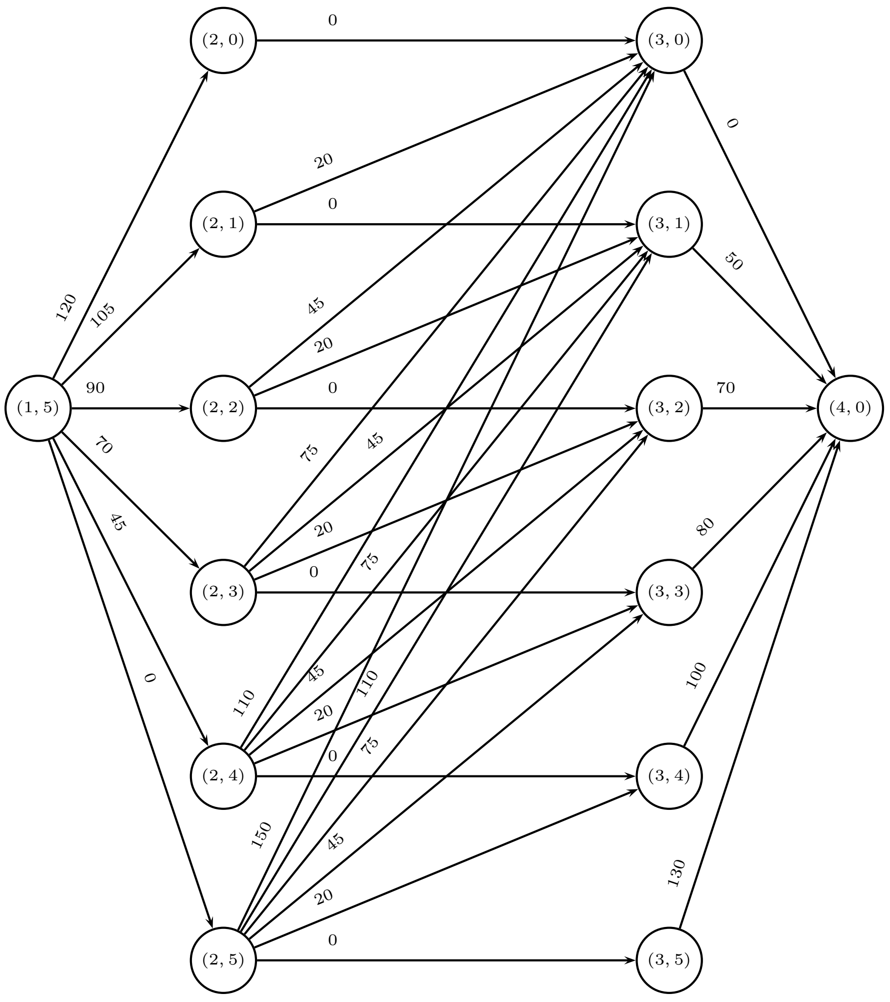


Figure 2.7: Network of states for Lothian Health Council.

The optimal value when we have s_n teams available at the start of stage n is

$$f_n^*(s_n) = \max\left\{f_n(s_n, x_n) \ / \ x_n \in \{0, 1, \ldots, s_n\}\right\}.$$

This means that we have the recursive expression

$$f_n(s_n, x_n) = p_n(x_n) + f_{n+1}^*(s_n - x_n).$$

Remember that we define $f_4^* = 0$.

As a consequence, the optimal values for each stage and state can be obtained with the following recursive formula:

$$f_n^*(s_n) = \max\left\{p_n(x_n) + f_{n+1}^*(s_n - x_n) \ / \ x_n \in \{0, 1, \ldots, s_n\}\right\}.$$

Let us use it now to solve the problem.

- *Stage $n = 3$.*
  Remember that $f_4^* = 0$. Thus:
  $f_3^*(s_3) = \max\{p_3(x_3)+f_4^*(s_3-x_3) \ / \ x_3 \in \{0, 1, \ldots, s_3\}\} = \max\{p_3(x_3) \ / \ x_3 \in \{0, 1, \ldots, s_3\}\}$.
  Let us compute the optimal value for the different states.

- $s_3 = 0$. $f_3^*(0) = \max\{p_3(0)\} = 0$. *Associated to* $x_3^* = 0$.
- $s_3 = 1$. $f_3^*(1) = \max\{p_3(0), p_3(1)\} = \max\{0, \underline{50}\} = 50$. *Associated to* $x_3^* = 1$.
- $s_3 = 2$. $f_3^*(2) = \max\{p_3(0), p_3(1), p_3(2)\} = \max\{0, 50, \underline{70}\} = 70$. *Associated to* $x_3^* = 2$.
- $s_3 = 3$. $f_3^*(3) = \max\{p_3(0), p_3(1), p_3(2), p_3(3)\} = \max\{0, 50, 70, \underline{80}\} = 80$. *Associated to* $x_3^* = 3$.
- $s_3 = 4$. $f_3^*(4) = \max\{p_3(0), p_3(1), p_3(2), p_3(3), p_3(4)\} = \max\{0, 50, 70, 80, \underline{100}\} = 100$. *Associated to* $x_3^* = 4$.
- $s_3 = 5$. $f_3^*(5) = \max\{p_3(0), p_3(1), p_3(2), p_3(3), p_3(4), p_3(5)\} = \max\{0, 50, 70, 80, 100, \underline{130}\} = 130$. *Associated to* $x_3^* = 5$.

- *Stage* $n = 2$.
  Here we have some more calculations because for each state there are several possible decisions: from not allocating any medical team at this stage to allocating all of them.

$$f_2^*(s_2) = \max\{p_2(x_2) + f_3^*(s_2 - x_2)\ /\ x_2 \in \{0, 1, \ldots, s_2\}\}.$$

  The optimal values for each state are:
  - $s_2 = 0$. $f_2^*(0) = \max\{p_2(x_2) + f_3^*(-x_2)\ /\ x_2 = 0\} = p_2(0) + f_3^*(0) = 0 + 0 = 0$. *Associated to* $x_2^* = 0$.
  - $s_2 = 1$. $f_2^*(1) = \max\{p_2(x_2) + f_3^*(1 - x_2)\ /\ x_2 \in \{0, 1\}\} = \max\{p_2(0) + f_3^*(1), p_2(1) + f_3^*(0)\} = \max\{0 + 50, 20 + 0\} = \max\{\underline{50}, 20\} = 50$. *Associated to* $x_2^* = 0$.
  - $s_2 = 2$. $f_2^*(2) = \max\{p_2(x_2) + f_3^*(2 - x_2)\ /\ x_2 \in \{0, 1, 2\}\} = \max\{p_2(0) + f_3^*(2), p_2(1) + f_3^*(1), p_2(2) + f_2^*(0)\} = \max\{0 + 70, 20 + 50, 45 + 0\} = \max\{\underline{70}, \underline{70}, 45\} = 70$. *Associated to* $x_2^* = 0$ *or* $x_2^* = 1$.
  - $s_2 = 3$. $f_2^*(3) = \max\{p_2(x_2) + f_3^*(3 - x_2)\ /\ x_2 \in \{0, 1, 2, 3\}\} = \max\{p_2(0) + f_3^*(3), p_2(1) + f_3^*(2), p_2(2) + f_3^*(1), p_2(3) + f_3^*(0)\} = \max\{0+80, 20+70, 45+50, 75+0\} = \max\{80, 90, \underline{95}, 75\} = 95$. *Associated to* $x_2^* = 2$.
  - $s_2 = 4$. $f_2^*(4) = \max\{p_2(x_2) + f_3^*(4 - x_2)\ /\ x_2 \in \{0, 1, 2, 3, 4\}\} = \max\{p_2(0) + f_3^*(4), p_2(1) + f_3^*(3), p_2(2) + f_3^*(2), p_2(3) + f_3^*(1), p_2(4) + f_3^*(0)\} = \max\{0 + 100, 20 + 80, 45 + 70, 75 + 50, 110 + 0\} = \max\{100, 100, 115, \underline{125}, 110\} = 125$. *Associated to* $x_2^* = 3$.
  - $s_2 = 5$. $f_2^*(5) = \max\{p_2(x_2) + f_3^*(5 - x_2)\ /\ x_2 \in \{0, 1, 2, 3, 4, 5\}\} = \max\{p_2(0) + f_3^*(5), p_2(1) + f_3^*(4), p_2(2) + f_3^*(3), p_2(3) + f_3^*(2), p_2(4) + f_3^*(1), p_2(5) + f_3^*(0)\} = \max\{0 + 130, 20+100, 45+80, 75+70, 110+50, 150+0\} = \max\{130, 120, 125, 145, \underline{160}, 150\} = 160$. *Associated to* $x_2^* = 4$.

- *Stage* $n = 1$.
  Finally, in this stage we only have to consider the state $s_1 = 5$*, as this is the number of available medical teams in our problem when we start.*

$$f_1^*(s_1) = \max\{p_1(x_1) + f_2^*(s_1 - x_1)\ /\ x_1 \in \{0, 1, \ldots, s_1\}\}.$$

  So, for $s_1 = 5$ *we have that:*
  $f_1^*(5) = \max\{p_1(x_1) + f_2^*(5 - x_1)\ /\ x_1 \in \{0, 1, 2, 3, 4, 5\}\} = \max\{p_1(0) + f_2^*(5), p_1(1) + f_2^*(4), p_1(2) + f_2^*(3), p_1(3) + f_2^*(2), p_1(4) + f_2^*(1), p_1(5) + f_2^*(0)\} = \max\{0 + 160, 45 + 125, 70 + 95, 90 + 70, 105 + 50, 120 + 0\} = \max\{160, \underline{170}, 165, 160, 155, 120\} = 170$. *Associated to* $x_1^* = 1$.

Therefore, the optimal allocation is:

- *1 medical team to country 1.*
- *3 medical teams to country 2.*
- *1 medical team to country 3.*

This will yield 170,000 additional person-years of life.

Note that here the order is not important because there is no actual time horizon. If we had chosen a different order for the countries, we would have obtained the same optimal value.

2.5. Dimension and the Curse of Dimensionality

From looking at the small size examples that we have seen in class we could think that it would be much easier to solve dynamic programming problems by total enumeration. Although this might be true for small size problems, we are going to see that this does not hold for bigger size problems.

Let us assume that our problem has $T+1$ stages with m states at each stage. Every decision causes a transition from a state on one stage to another stage on the next stage. Assume that there is one such transition for every pair of states on consecutive stages. See Figure 2.8.

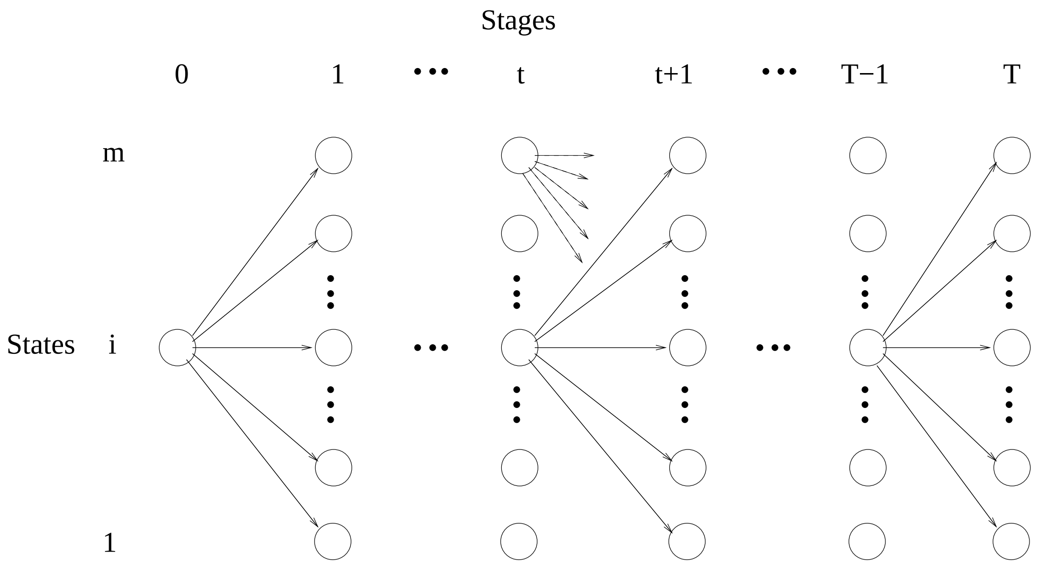


Figure 2.8: States growth.

Let us assume that the values for each state on the last stage we know without any calculation, and that we want to calculate the optimal value for stage 0.

<u>Method 1: Dynamic Programming.</u>

From the recursive expression

$$f_n(s, x_n) = d_{sx_n} + f^*_{n+1}(x_n),$$

we have that finding the optimal value for a state needs m additions and $m-1$ comparisons. So, $2m-1$ operations. We have $m(T-1)+1$ states that require this amount of calculations each. The $+1$ is to include the state at stage 0. Therefore, in total we need

$$(2m-1)\,(m(T-1)+1) = \mathcal{O}(m^2T)$$

operations. The $\mathcal{O}$ notation is used to take into account only the main element that determines approximately the number of operations (if you wish to know, the monomial with the highest degree).

Method 2: Total Enumeration.

In this case we need to evaluate the objective function for each path from the single state on stage 0 to each state on stage T. There are m arcs for each state to the states on the following stage. Therefore, the number of different paths is $m \times m \times \ldots \times m = m^T$. As evaluating each objective function takes T additions, we have $m^T T$ operations. We then have $m^T - 1$ comparisons, as the objective value for each path needs to be compared. In total we have

$$m^T T + m^T - 1 = \mathcal{O}(m^T T)$$

operations.

Therefore, the work required when using dynamic programming grows linearly on T, the number of stages, whereas for total enumeration it grows exponentially on T. If we assume that $m = 10$, then Table 2.13 shows how quickly one increases with regard the other.

	T				
	10	20	30	40	50
Dynamic Programming	1.7×10^3	3.6×10^3	5.5×10^3	7.4×10^3	9.3×10^3
Total Enumeration	1.1×10^{11}	2.1×10^{21}	3.1×10^{31}	4.1×10^{41}	5.1×10^{51}

Table 2.13: Effort of dynamic programming versus total enumeration.

The Curse of Dimensionality

Let us analyze now how the number of states at a stage depends on the dimension of the state space at that stage. Assume that the states at any stage are characterized by a vector $(y_1, y_2, \ldots, y_k)$, and that each component y_i can take r different values. For example, k different models of a pump, of which we can store any amount up to $r - 1$.

In this case the total number of states at one stage is r^k (the m with the previous notation). Therefore, the number of states grows exponentially with the dimension of the state space. For example, for $r = 10$ and $k = 20$ we have 10^{20} states at each stage.

We can summarize this section as follows:

- Dynamic programming behaves well (linearly) with the number of stages.
- Dynamic programming behaves badly (exponentially) with the dimension of the state space. This is known as the **curse of dimensionality**.

2.6. Probabilistic Dynamic Programming

In probabilistic dynamic programming the state at the next stage is not completely determined by the state and policy decision at the current stage. Instead, there is a probability distribution for what the next state will be. This distribution still is completely determined by the state and policy decision at the current state. The result structure can be seen in Figure 2.9.

We denote with S the number of possible states at stage $n+1$ and we label them $1, 2, \ldots, S$. The system goes to sate i with probability $p_i,\ i = 1, 2, \ldots, S$, given state s_n and decision x_n at stage n. If the system goes to state i, the contribution of stage n to the objective function is C_i.

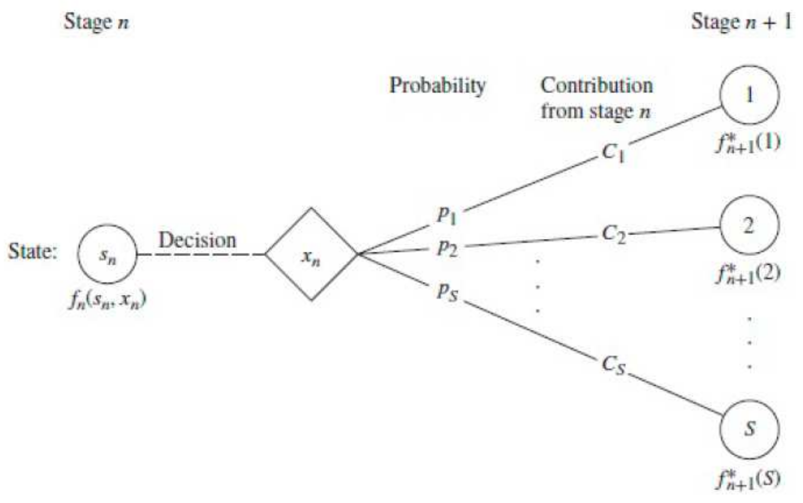


Figure 2.9: Probabilistic dynamic programming.

When this diagram is expanded to include all the possible states and decisions at all the stages, it is named a **decision tree**. It is a useful way of summarizing the information if the tree is not too large.

In this case the relationship between $f_n(s_n, x_n)$ and $f^*_{n+1}(s_{n+1})$ is more complicated and its precise form depends on the overall objective function.

For example, assume that the objective is to minimize the expected sum of the contributions from the individual stages. In this case $f_n(s_n, x_n)$ represents the minimum expected sum from stage n onward given that decision x_n is taken for state s_n at stage n. Therefore,

$$f_n(s_n, x_n) = \sum_{i=1}^{S} p_i \left(C_i + f^*_{n+1}(i)\right),$$

where

$$f^*_{n+1}(i) = \min \left\{f_{n+1}(i, x_{n+1})\right\}_{x_{n+1}},$$

with the range of the minimum considering only feasible values of x_{n+1}.

Let us see an example.

Example 2.7 (Determining Reject Allowances)
The company Edinburgh Wheels (EW) must supply one specific type of wheel to a customer. However, due to the high quality standards of this customer, EW may have to produce more than one wheel to obtain one that is acceptable. The number of extra items produced in a production run is called the reject allowance.

EW estimates that each of the wheels of this type that is produced will be acceptable with probability 0.5 *and defective (without any possibility for rework) with probability* 0.5. *This means that, for a lot of size L, the probability of producing exactly k acceptable wheels is* $\begin{pmatrix} L \\ k \end{pmatrix} 0.5^L$, *where* $\begin{pmatrix} L \\ k \end{pmatrix} = \frac{L!}{k!(L-k)!}$. *Particularly, the probability of producing no acceptable* *wheel on a run is* 0.5^L *(and the probability of producing at least one is equal to 1 minus that number).*

Production costs for this item are estimated to be £100 *(even if defective), and a maximum of 4 wheels can be produced per run. Excess items (if more than one acceptable wheel is produced on a given run) are worthless. In addition, a setup cost of £*300 *is incurred whenever the production process is set up for the product. A completely new setup at this cost is required for each production run that we decide to carry out. EW has time to make up to 3 production runs. If an acceptable item has not been obtained by the end of the third production run, the cost to EW in lost sales is £*$1,600$.

The manufacturer needs to choose a policy regarding the lot size (1 plus reject allowance) for the first production run, and then for each of the next two production runs if an acceptable item has not yet been produced. The objective is to determine the policy that minimizes the total expected cost.

<u>Formulation.</u>
We will use the following notation:

- *We represent with* n *each stage,* $n = 1, 2, 3$.
- x_n *is the lot size for stage* n.
- *The state* s_n *is the number of acceptable wheels still needed (0 or 1) when we begin stage* n.

Thus, $s_n = 1$ *at stage 1. Later,* $s_n = 0$ *or* $s_n = 1$*, depending on the success in producing an acceptable item before stage* n*. When at least one acceptable item is produced at stage* n*, then* $s_{n+1} = \ldots = s_n = 0$.

Now, because the objective that we need to minimize, we define:

- $f_n(s_n, x_n)$ *as the total expected cost for stages* $n, n+1, \ldots, 3$ *if the system starts in state* s_n *at stage* n*, the immediate decision is* x_n*, and optimal decisions are made thereafter.*
- $f_n^*(s_n) = \min\{f_n(s_n, x_n)\}_{x_n \in \{0,1,2,3,4\}}$.

We have that $f_n^*(0) = 0$ *for every stage* n *because there is no additional cost as soon as we do not need any more acceptable wheels.*

If we use £100 *as the monetary unit, the contribution from stage* n *to the total cost is* $K(x_n) + x_n$ *(regardless of the next state), where*

$$K(x_n) = \begin{cases} 0, & \text{if } x_n = 0, \\ 3, & \text{if } x_n \geq 1. \end{cases}$$

Therefore, for $s_n = 1$ *(when we still need at least one acceptable wheel) the cost will be the sum of the following parts:*

- *The setup cost.*
- *The production cost.*
- *The future minimal cost, which will depend on whether we manage to produce now at least one acceptable item. Thus:*
  - *If we do not produce it, then the future minimal cost is* $f_{n+1}^*(1)$*. This happens with probability* 0.5^{x_n}.
  - *If we produce it, then the future minimal cost is* $f_{n+1}^*(0) = 0$*. This happens with probability* $1 - 0.5^{x_n}$.

If we write this mathematically we have that

$$f_n(1,x_n) = K(x_n) + x_n + 0.5^{x_n} f^*_{n+1}(1) + (1-0.5^{x_n})\, f^*_{n+1}(0) = K(x_n) + x_n + 0.5^{x_n} f^*_{n+1}(1),$$

where $f^*_4(1) = 16$, the penalty for not producing any acceptable wheel in the process.

As a consequence, the recursive relationship is

$$f^*_n(1) = \min\left\{K(x_n) + x_n + 0.5^{x_n} f^*_{n+1}(1)\right\}_{x_n \in \{0,1,2,3,4\}},$$

with $n = 1, 2, 3$.

Solution.
Let us do the calculation for each of the three stages.

- *Stage $n = 3$.*

$$f^*_3(1) = \min\left\{K(x_3) + x_3 + 16 \cdot 0.5^{x_3}\right\}_{x_3 \in \{0,1,2,3,4\}}.$$

If $s_3 = 0$, then $f^*_3(0) = 0$ and $x^*_3 = 0$.

If $s_3 = 1$, then we have to compute the objective function for the different possible decisions and take the one with the smallest cost.

x_3	$f_3(1,x_3)$
0	$0+0+16 = 16$
1	$3+1+0.5 \times 16 = 12$
2	$3+2+0.5^2 \times 16 = 9$
3	$3+3+0.5^3 \times 16 = 8$
4	$3+4+0.5^4 \times 16 = 8$

Thus, $f^*_3(1) = 8$, which is associated to either $x^*_3 = 3$ or $x^*_3 = 4$.

- *Stage $n = 2$.*

$$f^*_2(1) = \min\left\{K(x_2) + x_2 + 8 \cdot 0.5^{x_2}\right\}_{x_2 \in \{0,1,2,3,4\}}.$$

The 8 inside the maximum is $f^*_3(1) = 8$, which we have calculate for stage $n = 3$.

If $s_2 = 0$, then $f^*_2(0) = 0$ and $x^*_2 = 0$.

If $s_2 = 1$, then we have to compute the objective function for the different possible decisions and take the one with the smallest cost.

x_2	$f_2(1,x_2)$
0	$0+0+8 = 8$
1	$3+1+0.5 \times 8 = 8$
2	$3+2+0.5^2 \times 8 = 7$
3	$3+3+0.5^3 \times 8 = 7$
4	$3+4+0.5^4 \times 8 = 7.5$

Thus, $f^*_2(1) = 7$, which is associated to either $x^*_2 = 2$ or $x^*_2 = 3$.

- *Stage* $n = 1$.
$$f_1^*(1) = \min \{K(x_1) + x_1 + 7 \cdot 0.5^{x_1}\}_{x_1 \in \{0,1,2,3,4\}} .$$
The 7 inside the maximum is $f_2^*(1) = 7$*, which we have calculate for stage* $n = 2$.

  Now $s_1 = 0$ *is not a possible state because we need to produce at least one acceptable wheel when we start the process.*

  For $s_1 = 1$*, then we have to compute the objective function for the different possible decisions and take the one with the smallest cost.*

x_1	$f_1(1, x_1)$
0	$0 + 0 + 7 = 7$
1	$3 + 1 + 0.5 \times 7 = 7.5$
2	$3 + 2 + 0.5^2 \times 7 = 6.75$
3	$3 + 3 + 0.5^3 \times 7 = 6.875$
4	$3 + 4 + 0.5^4 \times 7 = 7.4375$

  Thus, $f_1^*(1) = 6.75$*, which is associated to* $x_1^* = 2$.

The minimum total expected cost is $6.75 \times £100 = £675$.

The optimal policy is:

1. *Produce 2 wheels on the first production run.*
2. *If none is acceptable, produce either 2 or 3 wheels on the second production run.*
3. *If none is acceptable, produce either 3 or 4 wheels on the second production run.*

Chapter 3

Integer Programming

In many applications the solution to an optimization problem is restricted to take integer values. For example, the number of people allocated to work on a project or the number of cars that will be produced. In other occasions, the fact that we need to use integer variables is less obvious. For example, when modelling the condition that cars of model A can only be produced if at least 100 cars of model B are also produced.

Here we will study **mixed integer programming (MIP)** models, which means that some variables of the model are required to take integer values and some can take continuous values. When all the variables are restricted to take integer values, we speak of **pure integer programming**. However, unless it is important to differentiate, we simply speak in general of integer programming.

Particularly relevant are integer variables restricted to take values 0 or 1. They are called **binary variables** or 0-1 variables.

3.1. Rounding

One could think that integer programming problems are easy to solve. That we could solve the problem as a continuous problem and, if some of the variables required to be integer are fractional, then we could round to the closest integer because that should be the optimal solution. Unfortunately, although this may work sometimes, there is no guarantee that in general it will. Not just to obtain an optimal solution but to even obtain feasible solutions to the integer problem. The following example illustrates this.

Example 3.1 (Rounding Does Not Work)
Consider the problem

$$\begin{array}{ll} \textit{Max.} & x_2 \\ \textit{s.t.} & x_1 - 0.009x_2 \geq 1, \\ & x_1 \leq 1.9, \\ & x_1, x_2 \in \mathbb{Z}^+. \end{array}$$

The optimal solution to the problem where we drop the condition that the variables must take *integer values is shown in Figure 3.1 by the red point:* $(\tilde{x}_1, \tilde{x}_2) = (1.9, 100)$. *The optimal value is* $z_L^* = 100$.

If we round to the next integer value, we obtain the point $(x_1, x_2) = (2, 100)$, *the orange point in the figure. However, this point is not a feasible solution.*

Actually, the optimal solution to our problem is the green point: $(x_1^*, x_2^*) = (1, 0)$. *The optimal value is* $z_I^* = 0$. *As can be seen, rounding gives a completely wrong solution, not even close to the optimal solution.*

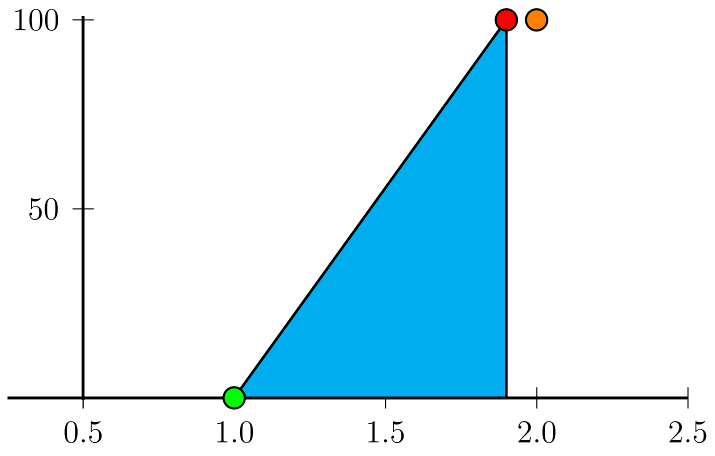


Figure 3.1: Rounding to the closest integer

3.2. A First Example and Some Applications

Example 3.2 (Location of factories)

The Alba Manufacturing Company (AMC) is considering expansion by building a new factory in either Glasgow or Aberdeen, maybe in both places. It is also considering building a new warehouse (but not two), but this is only possible in a city where a factory is also built. In addition, there is a certain number of employees required to work on supervising the construction of the buildings. AMC has 10 employees available for this project. The number of employees required and the net profitability (in tens of thousands of sterling pounds) of each of these decisions are on the table below.

	Net Profit		*Employees*	
Location	*Factory*	*Warehouse*	*Factory*	*Warehouse*
Glasgow	*9*	*6*	*6*	*5*
Aberdeen	*5*	*4*	*3*	*2*

What is the policy decision that maximizes the total net profit?

Solution:

First we name Glasgow as location 1 and Aberdeen as location 2. Using numbers will make our notation easier than if we use the city names.

Step 1: Decision variables.

Here we have a problem that involves yes/no decisions: to build or not to build. Thus, we *define the following decision variables:*

$$x_i = \begin{cases} 1, & \textit{if a factory is built on location } i, \\ 0, & \textit{otherwise.} \end{cases}$$

$$y_i = \begin{cases} 1, & \textit{if a warehouse is built on location } i, \\ 0, & \textit{otherwise.} \end{cases}$$

Step 2: Objective function.
Now that we have defined the decision variables, it is easy to write the objective function: we are maximizing the total net profit $9x_1 + 5x_2 + 6y_1 + 4y_2$.

Step 3: Constraints.
We need to write every requirement in the form as constraints. We must be careful because some are obvious, but some other are less evident and we might miss them if we do not pay attention.

We have a "budget" of employees. We cannot allocate more than those that are available:

$$6x_1 + 3x_2 + 5y_1 + 2y_2 \leq 10.$$

At most one warehouse can be built:

$$y_1 + y_2 \leq 1.$$

Finally, a warehouse can only be built where a factory is built:

$$\begin{aligned} y_1 &\leq x_1, \\ y_2 &\leq x_2. \end{aligned}$$

Let us analyze the first of these two constraints (the second is exactly the same but for the second location):

- *If* $y_1 = 1$*, then* $x_1 = 1$*. That is, if we build a warehouse on location 1, this is because we also build a factory on that location.*

  Equivalently, if $x_1 = 0$*, then* $y_1 = 0$*. That is, if there is not factory on location 1, then there cannot be a warehouse either.*

  Both things must happen for the constraint to be correctly written. A common mistake is to write constraints that include only some of the necessary conditions for the constraint to be correct. For example, if we had written $x_1 = y_1$*. Certainly, when* $y_1 = 1$*, then* $x_1 = 1$*, and when* $x_1 = 0$*, then* $y_1 = 0$*. But it does not allow the pair* $y_1 = 0$ *and* $x_1 = 1$*. That is, it is forcing to either build both buildings or none, but it does not allow for the decision of building a factory without building a warehouse, which is a valid decision according to the conditions of the problem.*
- *If* $y_1 = 0$*, then* x_1 *can take value either 0 or 1. The constraint does not impose any limitation. If we do not build a warehouse, then building or not a factory is irrelevant for the warehouse.*
- *The following table summarizes the different possibilities for the values of* x_1 *and* y_1 *and the outcome of the inequality representing the constraint:*

x_1	y_1	$y_1 \leq x_1$?
0	*0*	*Yes*
0	*1*	*No*
1	*0*	*Yes*
1	*1*	*Yes*

All the four possibilities must happen on the inequality as described on the table for the inequality to be correct.

Still regarding writing the constraints, we should not forget to impose that the variables are binary:

$$x_1, x_2, y_1, y_2 \in \{0, 1\}.$$

The Model.
Finally, we write all this information together as a mathematical optimization model.

$$\begin{array}{ll} \textit{Max.} & 9x_1 + 5x_2 + 6y_1 + 4y_4 \\ \textit{s.t.} & 6x_1 + 3x_2 + 5y_1 + 2y_4 \leq 10, \\ & y_1 + y_2 \leq 1, \\ & y_1 \leq x_1, \\ & y_2 \leq x_2, \\ & x_1, x_2, y_1, y_2 \in \{0, 1\}. \end{array}$$

In the previous example we have used binary variables to answer whether we should build a factory, a warehouse, or both. Other decisions that we can model with binary variables are:

- In finance, should we make a certain investment?
- In sales districting, should a certain distribution center be assigned to a certain market sector?
- In goods delivery with trucks, should a certain route be used?
- In exam timetabling, should exam A be scheduled before exam B?

3.3. Crew Scheduling and Covering Problems

One important area of applications of integer programming is aviation. The airline industry faces many complex problems that need to be solved with integer optimization techniques. One of the most important ones is the **crew scheduling problem**. It is necessary to assign sequences of flight legs to crews of pilots and flight attendants. The basic question that is modelled is "should a certain sequence of flight legs be allocated to a crew?" The objective is to minimize the total cost of scheduling crews that cover each flight leg of the schedule. Let us see an example.

Example 3.3 (Crew Scheduling)
Scottish Airlines (SA) needs to assign 3 crews to cover all its upcoming flights. The crews are based in Edinburgh and need to be allocated to the flights listed in the first column of Table 3.1. The other 12 columns show the 12 feasible sequences of flights for a crew, with the numbers indicating the order of the flights.

Exactly 3 of the sequences need to be chosen (one per crew) in such a way that every flight is covered. The cost of assigning a crew to a particular sequence of flights is given in thousands of sterling pounds. It is permissible to have more than one crew on a flight, where the extra *crews fly as passengers, but union contracts require that the extra crews are still paid for their time as if they are working. The objective is to minimize the total cost of the three crew assignments that cover all the flights.*

	Feasible Sequence of Flights											
Flight	1	2	3	4	5	6	7	8	9	10	11	12
1. Edinburgh to London	1			1			1			1		
2. Edinburgh to Paris		1			1			1			1	
3. Edinburgh to Dublin			1			1			1			1
4. London to Madrid				2			2		3	2		3
5. London to Edinburgh	2					3				5	5	
6. Madrid to Paris				3	3				4			
7. Madrid to Dublin							3	3		3	3	4
8. Paris to Edinburgh		2		4	4				5			
9. Paris to Madrid					2			2			2	
10. Dublin to Edinburgh			2				4	4				5
11. Dublin to London						2			2	4	4	2
Cost ($\times$£$1,000$)	2	3	4	6	7	5	7	8	9	9	8	9

Table 3.1: Flights and costs.

Solution.
We start by defining the following binary variables:

$$x_i = \begin{cases} 1, & \textit{is sequence } i \textit{ is assigned to a crew,} \\ 0, & \textit{otherwise} \end{cases}$$

The objective function that we seek to minimize is

$$2x_1 + 3x_2 + 4x_3 + 6x_4 + 7x_5 + 5x_6 + 7x_7 + 8x_8 + 9x_9 + 9x_{10} + 8x_{11} + 9x_{12}.$$

Now, when it comes to the constraints, we must cover every flight. Thus, for any flight we need to choose at least one sequence that include this flight.

In the case of the first flight, there are four sequences that cover this flights. The corresponding constraint is

$$x_1 + x_4 + x_7 + x_{10} \geq 1.$$

In the same way we write the constraints for the other 10 flights:

- *Flight 2:* $x_2 + x_5 + x_8 + x_{11} \geq 1.$
- *Flight 3:* $x_3 + x_6 + x_9 + x_{12} \geq 1.$
- *Flight 4:* $x_4 + x_7 + x_9 + x_{10} + x_{12} \geq 1.$
- *Flight 5:* $x_1 + x_6 + x_{10} + x_{11} \geq 1.$
- *Flight 6:* $x_4 + x_5 + x_9 \geq 1.$
- *Flight 7:* $x_7 + x_8 + x_{10} + x_{11} + x_{12} \geq 1.$
- *Flight 8:* $x_2 + x_4 + x_5 + x_9 \geq 1.$
- *Flight 9:* $x_5 + x_8 + x_{11} \geq 1.$
- *Flight 10:* $x_3 + x_7 + x_8 + x_{12} \geq 1.$

- *Flight 11:* $x_6 + x_9 + x_{10} + x_{11} + x_{12} \geq 1$.

There is also the requirement to assign 3 crews:

$$\sum_{i=1}^{12} x_i = 3.$$

Finally, we need to impose that all the variables are binary:

$$x_i\{0,1\},\ i = 1, 2, \ldots, 12.$$

All together (objective and constraints) are the model whose function we seek to minimize.

An optimal solution (that we can obtain with an optimization solver) is $x_3 = x_4 = x_{11} = 1$ *and* $x_i = 0$ *for all the other variables, with a total cost of* £$18,000$.

This crew scheduling example illustrates a broader category of problems name **set covering problems**. These are problems where we have to do a series of activities to provide some services at minimum cost. If S_j is the set of activities that can provide service j, then we have the following constraint:

$$\sum_{i \in S_j} x_i \geq 1.$$

Related to set covering problems we have **set partitioning problems**, whose characteristic constraint is of the form

$$\sum_{i \in S_j} x_i = 1.$$

Now, instead of having to choose a least one activity to perform the service, we need to choose exactly one.

3.4. Setup Costs

In many activities it is quite common to have some fixed charge or setup cost when performing an activity in the following way:

- If we do nothing, there is no setup cost.
- If we do the activity at some level, no matter how much, we have to pay a fixed setup cost in addition to any cost associated to the level of activity.

For example, this happens in a production chain if we think of the cost of preparing the different tools and machines of the chain for the production.

Usually the variable cost can be modelled as proportional to the level of activity, in which case we can represent the total activity cost as follows:

$$f(x) = \begin{cases} k + cx, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$$

Here $x \geq 0$ measures the level of activity, $k > 0$ is the setup cost, and $c > 0$ is the unit production cost. Note that here x can take noninteger values (for example, liters of petrol that are produced).

Without the presence of k, linear programming is enough to model this cost. However, with the presence of setup cost we need to use mixed integer linear programming. The key is to notice that, in order to decide whether we incur on the setup cost, we need to keep track on whether there is some level of activity or not. Thus, we need an auxiliary binary variable y that is defined as follows:

$$y = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, the objective function to be minimized is $g(x, y) = cx + ky$.

Now, note that the value of y depends on the value of x. In order for y to be well defined, we need to enforce it with one or more constraints. In this case, let us assume that M is a large positive number, greater than or equal to the maximum value that x can take (an upper bound). Then the constraint

$$x \leq My$$

ensures that y is well defined. Let us see it:

- If $x = 1$, then the constraint forces that $y = 1$. In addition, because $M \geq \max\{x\}_x$, we make sure that we do not leave out any feasible solution.
- Now, the only potential problem happens if $x = 0$. Then the constraints becomes $0 \leq My$, which means that y can take value either 0 or 1. However, if we look at the objective function, we see that it includes the term ky with $k > 0$. Therefore, since y is on no other constraint apart from $x \leq My$ and because we are minimizing, it is not optimal to choose $y = 1$. So, if $x = 0$, then $y = 0$.

Let us see an example.

Example 3.4 (Controlling Air Pollution)
Aberdeen Steel in Business (ASiB), a major producer of steel, is concerned about the pollution of its main factory. After conversations with a local environmental group, they have agreed some minimum levels for the reduction of emissions to the air. The three main types of pollutants are particulate matter, sulfur oxides, and hydrocarbons. Their emissions must be reduced yearly by at least 60,150, and 125 million kilograms, respectively.

There are two primary sources of pollution: blast furnaces and open-hearth furnaces. Three are the potential solutions for each of them: increasing the height of the smokestacks, using filter devices, and including cleaner materials in the fuels. Each of these methods has a technological limit on how much it can contribute to reduce pollution, but a fraction of this maximum can be used instead of full power, and different methods can be combined There is also a cost associated to the maximum feasible use and any fractional use below the maximum has a cost equal to that fraction of the full cost. Table 3.2 shows the reductions obtained with the different methods and Table 3.3 shows the costs. In addition, no matter the height increase of the smokestacks, there is a fixed cost of £2 millions if it is decided to modify them (per type of facility, so £4 millions if this technology is used for both types of furnaces).

What is the cheapest way to reduce the pollutant emissions by at least the levels that have been agreed?

<u>*Solution:*</u>
Let us start with the notation and the variables.

1. Notation and variables.

	Taller Smokestacks		Filters		Better Fuels	
Pollutant	Blast Furnaces	Open-Hearth Furnaces	Blast Furnaces	Open-Hearth Furnaces	Blast Furnaces	Open-Hearth Furnaces
Particulates	12	9	25	20	17	13
Sulfur Oxides	35	42	18	31	56	49
Hydrocarbonds	37	53	28	24	29	20

Table 3.2: Reductions (in millions of kilograms per year).

Technology	Blast Furnaces	Open-Hearth Furnaces
Taller Smokestacks	8	10
Filters	7	6
Better Fuels	11	9

Table 3.3: Cost for maximum feasible use (in millions of sterling pounds per year).

Let $I = \{1, 2, 3\}$ be the set of technologies that can used, where 1 is taller smokestacks, 2 is filters, and 3 is better fuels.

Let $J = \{1, 2\}$ be the set of facilities, where 1 is blast furnaces and 2 is open-hearth furnaces.

Although we will not use it here, it could be useful for some larger problems. Let us define anyway $K = \{1, 2, 3\}$ as the set of pollutants, where 1 is particulates, 2 is sulfur oxides, and 3 is hydrocarbons.

We define variables $x_{ij} \geq 0,\ i \in I,\ j \in J$, as the fraction of the maximum feasible use of technology i on facilities of type j.

In addition, as we have identified that there are setup costs, we define

$$y_j = \begin{cases} 1, & \textit{if taller smokestacks are implemented for furnaces of type } j, \\ 0, & \textit{otherwise,} \end{cases}$$

$j \in J$.

2. *Objective function.*

   The total cost is the sum of cost from the diverse technologies (fraction over the maximum feasible use) plus the setup cost.

   Thus, the objective function to be minimized is

$$8x_{11} + 10x_{12} + 7x_{21} + 6x_{22} + 11x_{31} + 9x_{32} + 2y_1 + 2y_2.$$

3. *Constraints.*

   We have several types of constraints.

   First of all, let us not miss that x_{ij} is a fraction of the maximum level of feasible use, which is represented as 1. Thus, we need to include the following constraints:

$$0 \leq x_{ij} \leq 1,\ i \in I,\ j \in J.$$

   There is a minimum level of reduction that must be met for each of the three pollutants:

- *Particulates:*
$$12x_{11} + 9x_{12} + 25x_{21} + 20x_{22} + 17x_{31} + 13x_{32} \geq 60.$$
- *Sulfur oxides:*
$$35x_{11} + 42x_{12} + 18x_{21} + 31x_{22} + 56x_{31} + 49x_{32} \geq 150.$$
- *Hydrocarbons:*
$$37x_{11} + 53x_{12} + 28x_{12} + 24x_{22} + 29x_{31} + 20x_{32} \geq 125.$$

Finally, we have the setup cost constraints. Since $x_{11} \leq 1$ and $x_{12} \leq 1$ we can take $M = 1$ in both cases. The constraints are

$$\begin{aligned} &x_{11} \leq y_1, \\ &x_{12} \leq y_2, \\ &y_1, y_2 \in \{0, 1\}. \end{aligned}$$

3.5. Logical Constraints

When working on an optimization model we may come across some yes/no decisions that are interrelated. It is what we call **logical constraints**. Let us think in terms of jobs that need to be done to see a nonexhaustive list of logical constraints. Given job j, we define variable x_j as follows:

$$x_j = \begin{cases} 1, & \text{if job } j \text{ is done,} \\ 0, & \text{otherwise.} \end{cases}$$

Some possible situations are discussed next.

1. If job 1 is done, then job 2 is also done.

   We model this as
   $$x_1 \leq x_2.$$
   If $x_1 = 1$, then $x_2 = 1$.

   The equivalent statement above is "if job 2 is not done, then job 1 is not done either". We can see that the inequality also satisfies this requirement: if $x_2 = 0$, then $x_1 = 0$.
2. If job 1 is done, then job 2 is not done.

   We can see this case in several different but equivalent ways:

   - We can simply write a constraint as above directly by considering the direct implications of the constraint:
   $$x_1 \leq 1 - x_2.$$
   - We can see the statement as "at most one of the two jobs can be done". This is so because, if job 2 is done, then it is not possible that we have done job 1 (otherwise, we would have done job 2, which is a contradiction). Thus:
   $$x_1 + x_2 \leq 1.$$

- We can also consider a new job $\tilde{2}$ that is "do not do job 2". Then are modelling "if job 1 is done, then job $\tilde{2}$ is also done". This is the first case that we studied. So, the constraint is
$$x_1 \leq x_{\tilde{2}}.$$
But it is clear from the definition of job $\tilde{2}$ that $x_{\tilde{2}} = 1 - x_2$. Thus, when we substitute we have that
$$x_1 \leq 1 - x_2.$$
Note that it is more usual the notation $\tilde{x}_2$ or $\bar{x}_2$ rather than $x_{\tilde{2}}$. We will not use again $\tilde{2}$ in our notation.

Let us see now an example that includes the use of logical constraints.

Example 3.5 (NASA Needs You)
NASA must decide how to split their budget among several spatial missions along the next five scientific periods. If a mission is undertaken, there is a cost that must be paid over one or more periods. There is a limited budget per period. In addition, it may happen that:

- *Two missions are incompatible (Inc): only one of them can be done.*
- *A mission needs of another mission (Only If): the mission can only be done if the indicated mission has also been done.*

Given that each mission has a certain scientific benefit (as a percentage of the whole study), write a model to find an optimal strategy (largest accomplished percentage).

	Cost (millions of dollars)							
Mission	*2022/ 2023*	*2024/ 2025*	*2026/ 2027*	*2028/ 2029*	*2030/ 2031*	*%*	*Inc*	*Only If*
(1) Communications satellite	*6*	–	–	–	–	*13*	–	–
(2) Orbital microwave	*2*	*3*	–	–	–	*2*	–	–
(3) Io lander	*3*	*3*	–	–	–	*4*	–	–
(4) Uranus orbiter 2028	–	–	–	*9*	*10*	*8*	*5*	*3*
(5) Uranus orbiter 2024	–	*5*	*8*	–	–	*6*	*4*	*3*
(6) Mercury probe	–	–	*1*	*8*	*4*	*2*	–	*3*
(7) Saturn probe	*1*	*8*	–	–	–	*4*	–	*3*
(8) Infrared imaging	–	–	–	*5*	–	*3*	*11*	–
(9) Ground-based SETI	*4*	*5*	–	–	–	*13*	*13, 14*	–
(10) Large orbital structures	–	*8*	*1*	–	–	*11*	–	–
(11) Color imaging	–	–	*2*	*7*	–	*4*	*8*	*2*
(12) Medical technology	*5*	*7*	–	–	–	*3*	–	–
(13) Polar orbital platform	–	*1*	*4*	*1*	*1*	*15*	*9,14*	–
(14) Geosynchronous SETI	–	*4*	*5*	*3*	*3*	*12*	*9,13*	–
Budget	*10*	*12*	*14*	*14*	*14*			

Solution.
We define variables
$$x_i = \begin{cases} 1, & \textit{if mission } i \textit{ is done,} \\ 0, & \textit{otherwise,} \end{cases}$$
$i = 1, \ldots, 14.$

The objective function to be maximized is

$$\begin{aligned}&13x_1 + 2x_2 + 4x_3 + 8x_4 + 6x_5 + 2x_6 + 4x_7 + 3x_8 + 13x_9 +\\ &11x_{10} + 4x_{11} + 3x_{12} + 15x_{13} + 12x_{14}.\end{aligned}$$

Now we have several constraints:

- *Budget constraints for each period:*

$$\begin{aligned}&6x_1 + 2x_2 + 3x_3 + x_7 + 4x_9 + 5x_{12} \leq 10,\\ &3x_2 + 3x_3 + 5x_5 + 8x_7 + 5x_9 + 8x_{10} + 7x_{12} + x_{13} + 4x_{14} \leq 12,\\ &8x_5 + x_6 + x_{10} + 2x_{11} + 4x_{13} + 5x_{14} \leq 14,\\ &9x_4 + 8x_6 + 5x_8 + 7x_{11} + x_{13} + 3x_{14} \leq 14,\\ &10x_4 + 4x_6 + x_{13} + 3x_{14} \leq 14.\end{aligned}$$

- *Incompatibility constraints:*

$$\begin{aligned}&x_4 + x_5 \leq 1,\\ &x_8 + x_{11} \leq 1,\\ &x_9 + x_{13} + x_{14} \leq 1.\end{aligned}$$

- *Dependency constraints:*

$$\begin{aligned}&x_4 \leq x_3,\\ &x_5 \leq x_3,\\ &x_6 \leq x_3,\\ &x_7 \leq x_3,\\ &x_{11} \leq x_2.\end{aligned}$$

- *We should not forget that our variables are binary:*

$$x_i \in \{0,1\},\ i = 1,\dots,14.$$

3.5.1. Either-Or Constraints

Something that we usually face in mathematical optimization problems is when we are given two constraints:

$$f(x_1,\dots,x_n) \leq 0, \tag{3.1}$$
$$g(x_1,\dots,x_n) \leq 0. \tag{3.2}$$

Then we are asked that at least one of the two inequalities is satisfied. This is called an **either-or constraint**.

In order to model that requirement we use integer programming. We introduce an auxiliary binary variable y that we define as follows:

$$y = \begin{cases} 0, & \text{if the first constraint is forced to be true,}\\ 1, & \text{if the second constraint is forced to be true.}\end{cases}$$

Now, let M be such that $M \geq \max\{f(x_1,\dots,x_n)\}_{(x_1,\dots,x_n)}$ and $M \geq \max\{g(x_1,\dots,x_n)\}_{(x_1,\dots,x_n)}$. Then the following constraints guarantee the desired outcome:

$$\begin{aligned}&f(x_1,\dots,x_n) \leq My,\\ &g(x_1,\dots,x_n) \leq M(1-y).\end{aligned}$$

If $y = 0$, then $f(x_1, \dots, x_n) \leq 0$ and $g(x_1, \dots, x_n) \leq M$. Thus, 3.1 holds and 3.2 may or may not hold. If $y = 1$, we have the same result but now we swap the roles of f and g.

Note: Instead of using M for both constraints, we can use $M_1 \geq \max\{f(x_1, \dots, x_n)\}_{(x_1,\dots,x_n)}$ and $M_2 \geq \max\{g(x_1, \dots, x_n)\}_{(x_1,\dots,x_n)}$. Both options are correct. The advantages of one or the other are technical and will not be discussed here.

Example 3.6 (Either-Or Constraint)
Dundee Autos is considering manufactguring three types of autos: compact, midsize, and large. The resources required for and the profits yielded by each type of car are shown on Table 3.4. Currently, 6,000 tons of steel and 60,000 hours of labor are available. For the production of a type of car to be economically feasible, at least 1,000 cars of that type must be produced. What is the best profit that the company can make?

	Car Type		
Resource	Compact	Midsize	Large
Steel required (in tons)	1.5	3	5
Labor required (in hours)	30	25	40
Profit yielded (£)	2000	3000	4000

Table 3.4: Resources and Profits.

Solution.
Dundee Autos needs to decide how many cars of each type to produce. We need thus three decision variables. We define x_i as the number of cars of type i that are produced, where 1 is compact, 2 is midsize, and 3 is large. These are nonnegative integer variables.

The objective function to be maximized is the total profit: $2x_1 + 3x_2 + 4x_3$. We are using thousands of sterling pounds as the monetary unit.

We have two very straightforward constraints:

- *There is a limit on the amount of steel that is available:*

$$1.5x_1 + 3x_2 + 5x_3 \leq 6000.$$

- *There is a limit on the hours of labor that are available i:*

$$30x_1 + 25x_2 + 40x_3 \leq 60000.$$

Now we need to write constraints for the condition regarding the minimum level of production. Let us consider cars of type 1. Types 2 and 3 are analogous.

We either do not produce anything ($x_1 \leq 0$), or we produce at least 1,000 cars ($x_1 \geq 1000$).

Clearly this is an either-or constraint. Thus, we define an auxiliar binary variable y_1 and add constraints

$$x_1 \leq M_1 y_1,$$
$$1000 - x_1 \leq M_1(1 - y_1).$$

We need to determine a value for M_1, which is easy if we analyze the resource constraints.

If $x_2 = x_3 = 0$ (their lowest possible values), those constraints become

$$\begin{aligned} 1.5x_1 &\leq 6000, \\ 30x_1 &\leq 60000. \end{aligned}$$

Therefore, $x_1 \leq \min\left\{\frac{6000}{1.5}, \frac{60000}{30}\right\} = \min\{4000, 2000\} = 2000$. We can take $M_1 = 2000$ (or any larger value).

In a similar way for the other two types of cars, we have that $M_2 = 2000$ and that $M_3 = 1200$.

When we write all together we have the following model:

$$\begin{array}{ll} \text{Max.} & 2x_1 + 3x_2 + 4x_3 \\ \text{s.t.} & 1.5x_1 + 3x_2 + 5x_3 \leq 6000, \\ & 30x_1 + 25x_2 + 40x_3 \leq 6000, \\ & x_i \leq M_i y_i, \quad i = 1, 2, 3, \\ & 1000 - x_i \leq M_i(1 - y_i), \quad i = 1, 2, 3, \\ & x_i \in \mathbb{Z}^+, \quad i = 1, 2, 3, \\ & y_i \in \{0, 1\}, \quad i = 1, 2, 3, \end{array}$$

where $M_1 = M_2 = 2000$ and $M_3 = 1200$.

It is easy to extend the previous result to impose that at least k out of m constraints are satisfied. Assume that we have the conditions

$$f_i(x_1, \ldots, x_n) \leq 0, \; i = 1, \ldots, m.$$

We would like that at least k out of these m inequalities hold.

In order to achieve this, we introduce auxiliary binary variables y_i, $i = 1, \ldots, m$, and we add the following constraints to the model that we are solving:

$$\begin{aligned} & f_i(x_1, \ldots, x_n) \leq M_i(1 - y_i), \; i = 1, \ldots, m, \\ & y_1 + \ldots + y_m \geq k, \\ & \qquad\qquad y_i \in \{0, 1\}, \; i = 1, \ldots, m, \end{aligned}$$

where $M_i \geq \max\{f_i(x_1, \ldots, x_n)\}_{(x_1, \ldots, x_n)}$.

At least k variables y_i will have value 1, which means that at least k times we will have a constraint of the form $f_i(x_1, \ldots, x_n) \leq 0$.

3.5.2. If-Then Constraints

In some other occasions we need to guarantee that, if a constraint $f(x_1, \ldots, x_n) > 0$ is satisfied, then a second constraint $g(x_1, \ldots, x_n) \geq 0$ is also satisfied. If the first constraint is not satisfied, then we do not mind what happens with the second constraint. This is called an **if-then constraint**.

To achieve this, we create an auxiliary binary variable y and we include the following constraints:

$$\begin{aligned} f(x_1, \ldots, x_n) &\leq M(1 - y), \\ -g(x_1, \ldots, x_n) &\leq My, \end{aligned}$$

where $M \geq \max\{f(x_1,\ldots,x_n)\}_{(x_1,\ldots,x_n)}$ and $M \geq \max\{-g(x_1,\ldots,x_n)\}_{(x_1,\ldots,x_n)}$. The latter inequality is equivalent to $M \geq -\min\{g(x_1,\ldots,x_n)\}_{(x_1,\ldots,x_n)}$.

Note: As for the either-of constraints, instead of using M for both constraints, we can use $M_1 \geq \max\{f(x_1,\ldots,x_n)\}_{(x_1,\ldots,x_n)}$ and $M_2 \geq \max\{-g(x_1,\ldots,x_n)\}_{(x_1,\ldots,x_n)}$. Then we use M_1 instead of M on the first constraint and M_2 on the second. Both options are correct.

If $f(x_1,\ldots,x_n) > 0$, then necessarily $y = 0$. Therefore, $-g(x_1,\ldots,x_n) \leq 0$. That is, $g(x_1,\ldots,x_n) \geq 0$, which is what we want. On the other hand, if $f(x_1,\ldots,x_n) \leq 0$, then $y \in \{0,1\}$, which means that $-g(x_1,\ldots,x_n) \leq My \leq M$. Thus, nothing is enforced on the value of g.

Let us see an example.

Example 3.7 (If-Then Constraint)
Assume that we would like to model that, if $x_1 = 1$, then $x_2 = x_3 = x_4 = 0$. All of them are binary variables.

Because the variables are binary, the previous condition is equivalent to the following:

$$\text{If } x_1 > 0, \text{ then } x_2 + x_3 + x_4 \leq 0 \text{ (or } -x_2 - x_3 - x_4 \geq 0).$$

If we now define $f(x_1,\ldots,x_4) = x_1$ and $g(x_1,\ldots,x_4) = -x_2 - x_3 - x_4$, we can use the last studied result to write that the condition is satisfied if we include the following constraints:

$$\begin{aligned} x_2 + x_3 + x_4 &\leq M_2 y, \\ x_1 &\leq M_1(1-y), \\ y &\in \{0,1\}. \end{aligned}$$

In order to determine values for M_1 and M_2, we compute the following:

- $M_1 \geq \max\{f(x_1,\ldots,x_4)\}_{(x_1,\ldots,x_4)} = \max\{x_1\}_{(x_1,\ldots,x_4)} = 1.$
- $M_2 \geq \max\{-g(x_1,\ldots,x_4)\}_{(x_1,\ldots,x_4)} = \max\{x_2 + x_3 + x_4\}_{(x_1,\ldots,x_4)} = 3.$

Therefore, the constraints are

$$\begin{aligned} x_2 + x_3 + x_4 &\leq 3y, \\ x_1 &\leq 1 - y, \\ y &\in \{0,1\}. \end{aligned}$$

3.6. Piecewise Linear Functions

A **piecewise linear function** is a function that consists of several straight-line segments. See an example in Figure 3.2.

The points where the slope of the piecewise linear function changes (or the range of the definition of the function begins or ends) are called **break points**. Thus, in Figure 3.2, the breakpoints are 0,10,30,40, and 50.

These functions appear often in optimization problems when modelling aspects of an economy of scale. Buying a few items is not the same as buying many items.

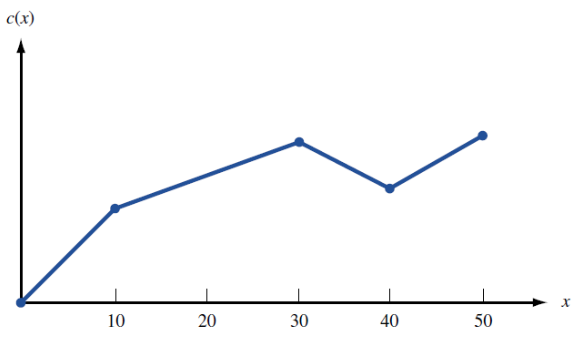


Figure 3.2: Piecewise linear function.

Example 3.8 (Buying Oil)
Suppose that we manufacture gasoline from oil. In purchasing oil from our supplier we receive a discount for large orders. The first 500 liters cost 25p per liter, the next 500 liters cost 20p per liter, and the next 500 liters cost 15p per liter. At most 1,500 liters can be purchased.

In order to model this cost function, let x be the number of liters of oil purchased, and let $c(x)$ be the cost in pennies of purchasing $c(x)$ liters of oil.

- *If $0 \leq x \leq 500$, then $c(x) = 25x$.*
- *If $500 \leq x \leq 1000$, then the cost is the cost of the first 500 liters (500×25) plus the cost of the liters in excess of 500. Thus, $c(x) = 500 \times 25 + 20(x - 500) = 20x + 2500$.*
- *Likewise, if $1000 \leq x \leq 1500$, we have that $c(x) = 500 \times 25 + 500 \times 20 + 15(x - 1000) = 15x + 7500$.*

Therefore, the cost function is

$$c(x) = \begin{cases} 25x, & 0 \leq x \leq 500, \\ 20x + 2500, & 500 \leq x \leq 1000, \\ 15x + 7500, & 1000 \leq x \leq 1500. \end{cases}$$

The break points are 0,500,1000, and 1500.

A piecewise linear function is not a linear function, but it can be represented in linear form with the help of binary variables.

Suppose that a piecewise linear function $f(x)$ has break points $b_1, b_2, \ldots, b_n$. For any point $\tilde{x}$ in the range of definition of f, there is $k \in \{1, 2 \ldots, n-1\}$ such that $b_k \leq \tilde{x} \leq b_{k+1}$.

Then, for some $\lambda_k \in [0, 1]$ we can write $\tilde{x}$ as a convex combination of the two break points:

$$\tilde{x} = \lambda_k b_k + (1 - \lambda_k) b_{k+1}.$$

Now, remember that f is linear in the interval $[b_k, b_{k+1}]$. Assume that $f(x) = c_k x + d_k$ in this interval. Then we have that

$$f(\tilde{x}) = c_k \tilde{x} + d_k = c_k \left(\lambda_k b_k + (1 - \lambda_k) b_{k+1}\right) + d_k.$$

We use that we can write $d_k = \lambda_k d_k + (1-\lambda_k)d_k$ and then we have that

$$f(\tilde{x}) = \lambda_k c_k b_k + (1-\lambda_k)c_k b_{k+1} + \lambda_k d_k + (1-\lambda_k)d_k = \lambda_k(c_k b_k + d_k) + (1-\lambda_k)(c_k b_{k+1} + d_k) = \lambda_k f(b_k) + (1-\lambda_k) f(b_{k+1}).$$

For example, in the oil problem, if $\tilde{x} = 800$, then $\tilde{x}$ is between break points $x_2 = 500$ and $x_3 = 1000$. So, now we need to solve

$$800 = 500\lambda + 1000(1-\lambda).$$

The solution is $\lambda = \frac{2}{5}$ and we have that:

- $\tilde{x} = \frac{2}{5} \times 500 + \frac{3}{5} \times 1000$.
- $f(\tilde{x}) = \frac{2}{5} \times f(500) + \frac{3}{5} \times f(1000) = 18500$.

However, in order to integrate this in the optimization model we see that we do not know which value $\tilde{x}$ will be used in the solution of the problem *before* solving the problem. Thus, we must allow the model to decide this. In order to do it we do the following:

1. Wherever $f(x)$ appears in the optimization model, replace it with

$$\lambda_1 f(b_1) + \lambda_2 f(b_2) + \ldots + \lambda_n f(b_n).$$

   Note that values $\{f(b_i)\}_{i=1}^n$ are known. Only values $\{\lambda_i\}_{i=1}^n$ are unknown. They will be decision variables.

2. We add the following constraints that enforce that any point x is a convex combination of two consecutive break points:

$$\begin{aligned}
b_1\lambda_1 + b_2\lambda_2 + \ldots + b_n\lambda_n &= x, && (3.3)\\
\lambda_1 + \lambda_2 + \ldots + \lambda_n &= 1, && (3.4)\\
\lambda_1 &\le \mu_1, && (3.5)\\
\lambda_2 &\le \mu_1 + \mu_2, &&\\
&\vdots &&\\
\lambda_{n-1} &\le \mu_{n-2} + \mu_{n-1}, &&\\
\lambda_n &\le \mu_{n-1}, && (3.6)\\
\mu_1 + \mu_2 + \ldots + \mu_{n-1} &= 1, && (3.7)\\
\mu_i &\in \{0,1\},\ i = 1,\ldots,n-1, && (3.8)\\
\lambda_i &\ge 0,\ i = 1,\ldots,n. && (3.9)
\end{aligned}$$

   The first two constraints, (3.3) and (3.4), plus (3.9) ensure that any point x is expressed as a convex combination of the break points.

   But remember that, since we are determining in which segment $[b_k, b_{k+1}]$ point x lies (because the definition of function f depends on this), we need to use only two break points and we need that they are consecutive. This is achieved by introducing the binary variables μ_i and the other constraints.

   Since $\mu_1 + \mu_2 + \ldots + \mu_{n-1} = 1$, exactly one variable μ_i will take value 1. All the other variables will take value 0, which forces that all the constraints (3.5)-(3.6) except two will have right-hand side with value 0. Thus, the corresponding λ_i variables take value 0.

Then exactly two constraints in (3.5)-(3.6) will have right-hand side with value 1. Thus, we have that $\lambda_i \leq 1$ and $\lambda_{i+1} \leq 1$ for some i.

Therefore, when that happens, constraints (3.3)-(3.9) are reduced to

$$\begin{aligned} b_i\lambda_i + b_{i+1}\lambda_{i+1} &= x, \\ \lambda_i + \lambda_{i+1} &= 1, \\ \lambda_i, \lambda_{i+1} &\geq 0, \end{aligned}$$

which is exactly what we want.

Let us see an example.

Example 3.9 (Gasoline Production)
Arran Gas produces two types of gasoline (gas 1 and gas 2) from two types of oil (oil 1 and oil 2). Each liter of gas 1 must contain at least 50 percent of oil 1, and each liter of gas 2 must contain at least 60 percent of oil 1. Each liter of gas 1 can be sold for 12p, and each liter of gas 2 can be sold for 14p. Currently, 500 liters of oil 1 and 1000 liters of oil 2 are available. As many as 1,500 more liters of oil 1 can be purchased at the following prices: 25p per liter for the first 500 liter, 20p per liter for the next 500 liters, and 15p per liter for the next 500 liters. Write a mathematical optimization model to maximize the profit of the company (revenues minus purchasing costs).

Solution:
This is a blending problem: some resources are combined to obtain some products. In addition, the cost function for purchasing extra oil 1 is a piecewise linear function.

We start by defining the following variables:

- *x is the amount (in liters) of oil 1 purchase.*
- *x_{ij} is the amount (in liters) of oil i used to produce gas j, $i, j = 1, 2$.*

The liters of gas 1 produced are $x_{11} + x_{21}$ because there is no loss of resources in the blending process (otherwise we would have been told). We could define a new variable $g_1 = x_{11} + x_{21}$ to keep the notation simpler, but we will not do it in this case. Likewise, the liters of gas 2 produced are $x_{12} + x_{22}$.

Then, the objective function to maximize is

$$\textit{total revenue - purchasing cost} = 12(x_{11} + x_{21}) + 14(x_{12} + x_{22}) - c(x),$$

where we had already determined in the previous example that

$$c(x) = \begin{cases} 25x, & 0 \leq x \leq 500, \\ 20x + 2500, & 500 \leq x \leq 1000, \\ 15x + 7500, & 1000 \leq x \leq 1500. \end{cases}$$

We write now the constraints:

- *Limit on oil 1 that can be used:*

$$x_{11} + x_{12} \leq 500 + x.$$

- *Limit on oil 2 that can be used:*

$$x_{21} + x_{22} \leq 1000.$$

- *The oil mixed to make gas 1 must be at least 50% oil 1:*

$$\frac{x_{11}}{x_{11} + x_{21}} \geq 0.5.$$

This constraint is nonlinear, but we can linearize it easily.

As $x_{11} + x_{21} \geq 0$, we multiply both sides of the constraint by that amount to obtain that (without changing the sign of the inequality):

$$\begin{aligned} x_{11} &\geq 0.5x_{11} + 0.5x_{21}; \\ 0.5x_{11} &\geq 0.5x_{21}; \\ x_{11} &\geq x_{21}. \end{aligned}$$

- *The oil mixed to make gas 2 must be at least 60% oil 1:*

$$\frac{x_{12}}{x_{12} + x_{22}} \geq 0.6.$$

As for gas 1, we linearize the constraint:

$$\begin{aligned} x_{12} &\geq 0.6x_{12} + 0.6x_{22}; \\ 0.4x_{12} &\geq 0.6x_{22}. \end{aligned}$$

- *All these variables are nonnegative, with variable x upperly bounded by $1,500$:*

$$\begin{aligned} x_{11}, x_{12}, x_{21}, x_{22} &\geq 0, \\ 0 \leq x &\leq 1500. \end{aligned}$$

- *Finally, we need to include the elements necessary to linearize the piecewise linear cost function. The break points are $b_1 = 0,\ b_2 = 500,\ b_3 = 1000,$ and $b_4 = 1500$. Therefore:*

  1. *We replace $c(x)$ with $c(0)\lambda_1 + c(500)\lambda_2 + c(1000)\lambda_3 + c(1500)\lambda_4$.*
  2. *We add the following constraints:*

$$\begin{aligned} 500\lambda_2 + 1000\lambda_3 + 1500\lambda_4 &= x, \\ \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 &= 1, \\ \lambda_1 &\leq \mu_1, \\ \lambda_2 &\leq \mu_1 + \mu_2, \\ \lambda_3 &\leq \mu_2 + \mu_3, \\ \lambda_4 &\leq \mu_3, \\ \mu_1 + \mu_2 + \mu_3 &= 1, \\ \mu_i &\in \{0,1\},\ i = 1,2,3, \\ \lambda_i &\geq 0,\ i = 1,2,3,4. \end{aligned}$$

3.7. Graphical Solution of Linear Problems

We are going to study later two fundamental methods to solve mixed integer linear problems: branch-and-bound and branch-and-cut. Both are based on the assumption that we know how to solve continuous linear problems. Usually we do that with an optimization solver: CPLEX, Gurobi, and Xpress are three widely used, but there are many other. Even Excel has an add-on that allows to solve linear problems. However, here instead we will work with problems with only two variables that we will solve graphically. This will allow us to familiarize with the different concepts that are key elements in (integer) linear programming that we would be unable to visualize otherwise.

In order to solve graphically a linear problem, we need to draw the feasible region and then we have to study the objective function. Thus, we start on the plane $\mathbb{R}^2$, and variables x_1 and x_2 will take values along the horizontal and vertical axes, respectively.

Next we delimit the feasible region P by drawing the constraints of the problem one by one: an equality ($x_1 + x_2 = 3$) is represented with a straight line, whereas an inequality ($x_1+x_2 \leq 3$) is a closed half-plane. Note that most of the times we will have the nonnegativity constraints $x_1, x_2 \geq 0$, which means that the feasible region is contained in the upper right quadrant.

Once we have drawn the feasible region, we consider the objective function, which is of the form $c_1x_1 + c_2x_2$. For each level straight line $z = c_1x_1 + c_2x_2$ we have a straight line, parallel to the other straight lines. The one that intersects the feasible region for the largest value z^* provides us with an optimal solution.

Let us see with an example how to solve graphically a linear problem step by step.

Example 3.10
Solve graphically the following continuous linear problem:

$$\begin{cases} \textit{Max.} & 5x_1 + 3x_2 \\ \textit{s.t.} & x_1 + x_2 \leq 7, \\ & 3x_1 + 4x_2 \leq 24, \\ & x_2 \leq 5, \\ & -6x_1 + x_2 \leq 1, \\ & x_1, x_2 \geq 0. \end{cases}$$

Solution:
First we draw the feasible region. We represent the inequalities one by one. As each of them is associated to a half-plane, the feasible region will become smaller and smaller because it is the intersection of all these half planes.

We start with the two nonnegativity constraints: $x_1, x_2 \geq 0$. See Figure 3.3.

Next we consider the straight line $x_1 + x_2 \leq 7$. In order to know which side is the "feasible side", we take one arbitrary point. Usually, the most convenient is $(0,0)$ (unless it is on the straight line). If the point satisfies the inequality, then the half-plane where that point lies is the one that is used to determine the feasible region. Otherwise, it is the other half-plane. See Figure 3.4.

The vertices of the feasible region are obtained as the intersection of the two straight lines

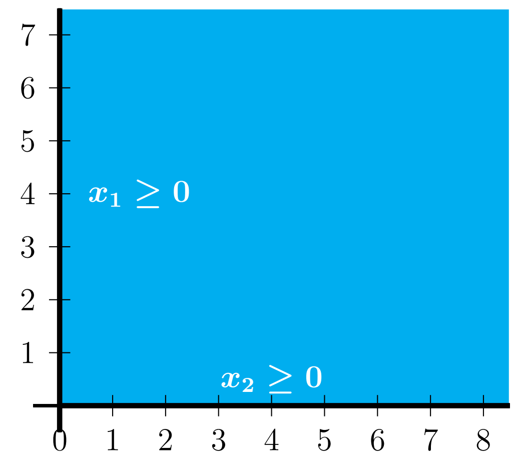


Figure 3.3: Drawing the feasible region. Step 1.

that cut at that point. In order to obtain them, we solve the system of two equations where the constraints are written as equalities.

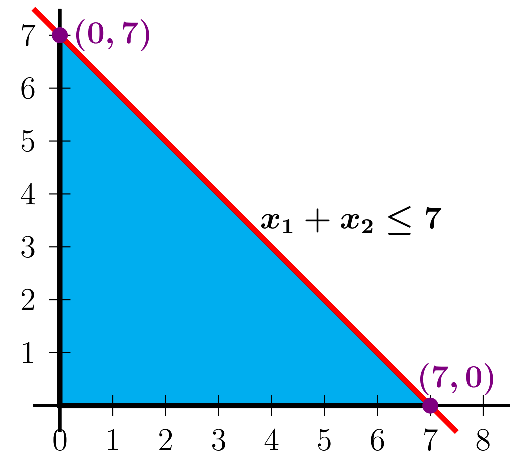


Figure 3.4: Drawing the feasible region. Step 2.

With the next constraint, $3x_1 + 4x_2 \leq 24$*, we see how the feasible region is even smaller. See Figure 3.5. The orange area has been cut off by this constraint.*

We do the same for constraint $x_2 \leq 5$ *(see Figure 3.6):*

And for $-6x_1 + x_2 \leq 1$ *(see Figure 3.7).*

Finally, the feasible region is obtained. See Figure 3.8.

Once we have the feasible region, we draw several level straight lines for the objective function. As we are maximizing, we see that we need to move to the right along the horizontal axis OX_1*. Particuarly, we see that the level straight line of maximum value that intersects the feasible region has value 35. See Figure3.9.*

Therefore, this problem has optimal value $z^* = 35$ *and the optimal solution is* $(x_1^*, x_2^*) = (7, 0)$

Note that it is only by chance that the optimal solution is integer. As we will see in the next example, in general, one or more of the variables will have a fractional value.

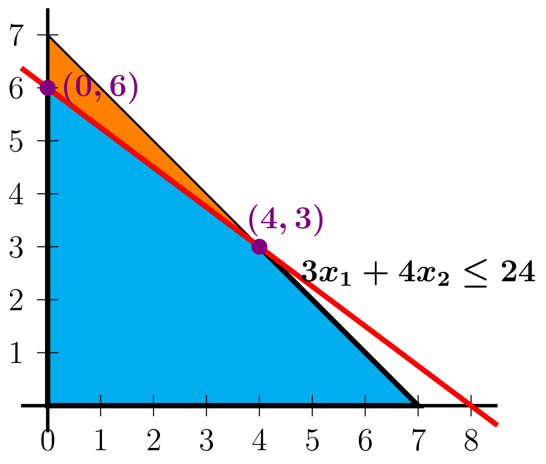


Figure 3.5: Drawing the feasible region. Step 3.

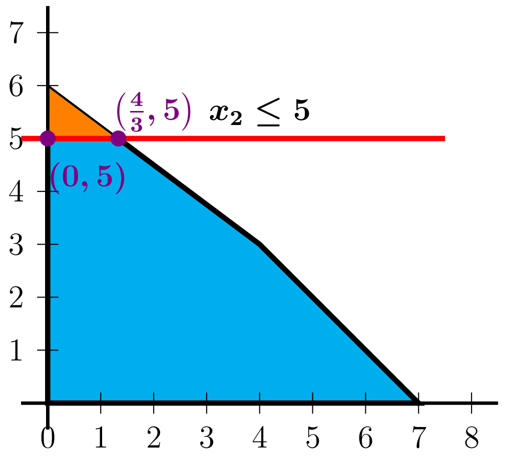


Figure 3.6: Drawing the feasible region. Step 4.

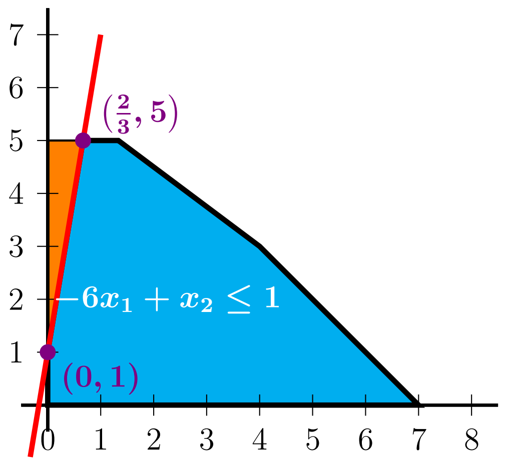


Figure 3.7: Drawing the feasible region. Step 5.

Example 3.11

If on the previous example we are maximizing function $-x_1 + 2x_2$ instead, the feasible region does not change because the constraints are the same. However, when we draw the level

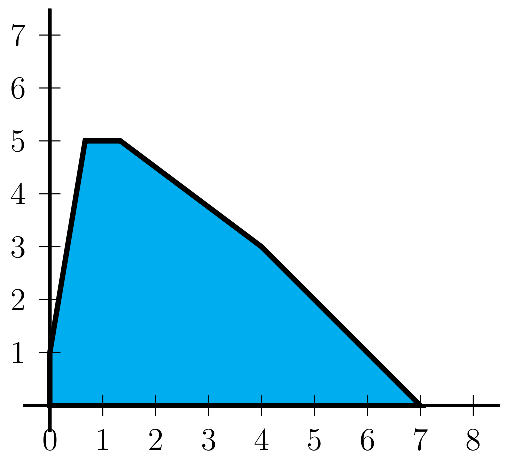


Figure 3.8: Drawing the feasible region. Step 6.

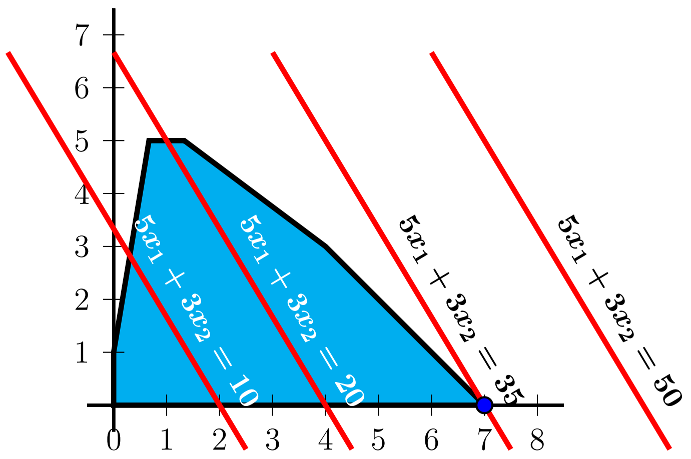


Figure 3.9: Drawing level straight lines for the objective function.

straight lines, we have a different optimal solution and value. See Figure 3.10.

The optimal value is $z^* = \frac{28}{3}$ *and the optimal solution is* $(x_1^*, x_2^*) = \left(\frac{2}{3}, 5\right)$.

The previous examples share one trait: the optimal solution is a vertex of the feasible region. This is not by chance. It is an important property of linear programming: if a linear problem has a bounded optimal solution, then there is a vertex of the feasible region that is an optimal solution. Therefore, when drawing the level straight lines, we look for the "last" vertex that a level of the objective function touches that makes its value as large as possible (if we are maximizing), or as small as possible (if we are minimizing).

3.8. The Branch-and-Bound Method

Branch-and-bound is the most popular method to solve integer programming problems. It solves a collection of continuous linear problems in order to solve the original integer linear problem. It is based on an elemental but key observation: if you solve the **linear relaxation** of an integer problem (the problem obtained by dropping the requirement that some of the variables need to be integer), and its solution has all variables with integer values, then that

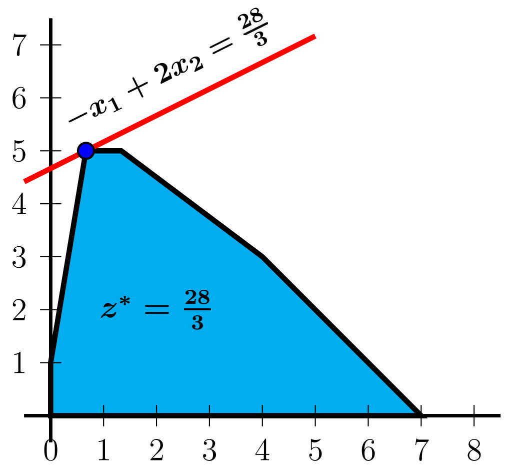


Figure 3.10: A different objective function.

optimal solution is also optimal to the integer problem. This is true because the feasible region of the linear relaxation contains and it is larger than the feasible region of its original integer problem.

Now, we will develop the elements of branch-and-bound through an example. The different linear problems that appear will be solved graphically. Noted that some details are omitted, like drawing the levels of the objective function for each of the subproblems that are solved, but that is part of what you would have to do if you were to solve the problem yourself.

Example 3.12 (Branch-and-Bound by Example)
The Lothian Corporation manufactures tables and chairs. A table requires 1 hour of labor and 9 square board feet of wood. A chair requires 1 hour of labor and 5 square board feet of wood. Currently, 6 hours of labor and 45 square board feet of wood are available. Each table contributes £8 to the profit and each chair contributes £5. Formulate and solve an integer problem to maximize the company's profit.
Solution:
Let x_1 be the number of tables manufactured and let x_2 be the number of chairs manufactured. Both decision variables are nonnegative and integer.

The problem that we need to solve is

$$\begin{array}{ll} \textit{Max.} & 8x_1 + 5x_2 \\ \textit{s.t.} & x_1 + x_2 \leq 6, \\ & 9x_1 + 5x_2 \leq 45, \\ & x_1, x_2 \in \mathbb{Z}^+. \end{array}$$

The branch-and-bound method begins by solving the linear relaxation of the integer problem. If we are lucky and all the decision variables required to be integer take integer values in the optimal solution that we obtain, then we have also solved the integer problem.

In this case, unfortunately, this does not happen. The optimal solution of the linear relaxation of the previous problem is $\tilde{x} = \left(\frac{15}{4}, \frac{9}{4}\right)$ with optimal value $z = \frac{165}{4}$. See Figure 3.11.

As we are maximizing, we know that $z_{IP} \leq z_{LP}$, where z_{IP} and z_{LP} are the optimal values of the integer problem and its linear relaxation, respectively. Therefore $z = \frac{165}{4}$ is an upper bound to the optimal value of the integer problem.

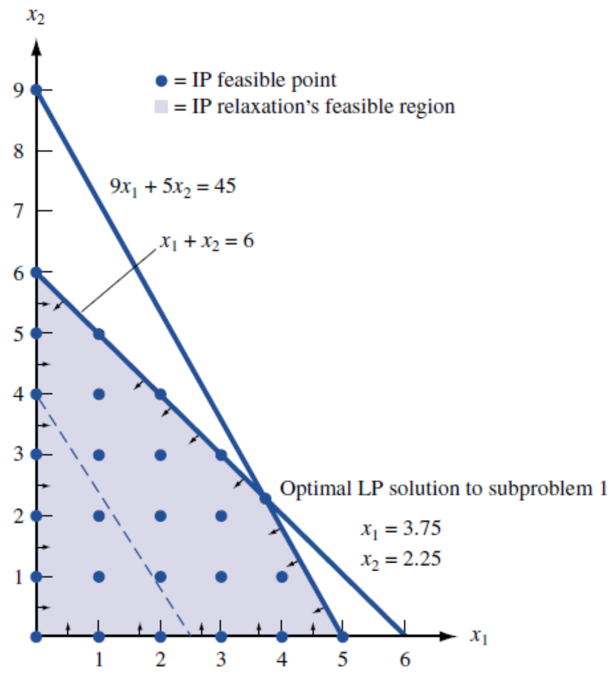


Figure 3.11: Feasible Region for Lothian Problem.

Next we are going to partition the feasible region for the linear relaxation. We are trying to find the region where an optimal solution is. In order to do that, we divide the feasible region into smaller parts (potentially discarding parts where there cannot be an optimal solution) and we explore them one by one.

In this case, we arbitrarily choose a variable that is fractional in the optimal solution to the linear problem. For example, x_1. We have obtained that $\tilde{x}_1 = \frac{15}{4} = 3.75$. As in the optimal solution to the integer problem x_1 must take an integer value, it must be either $x_1 \leq 3$ or $x_1 \geq 4$. Thus, we are going to create two new subproblems where one of those constraints is added to each of them. We say that we are branching *on that variable because this process is usually represented through a tree. The nodes of the tree contain the information about the linear relaxation solved and the branches show information about what new conditions we are adding to partition the problem of the node they leave from.*

If we name Subproblem 1 our original integer problem, we now have two new subproblems:

- *Subproblem 2: Subproblem 1 + constraint $x_1 \geq 4$.*
- *Subproblem 3: Subproblem 1 + constraint $x_1 \leq 3$.*

Observe that neither subproblem 2 nor subproblem 3 include point $\tilde{x}$ as a feasible point. This is done on purpose and it is one of the key elements of the branch-and-bound method: the new subproblem created (the children nodes on the branching tree) cut off the optimal solution of the linear relaxation of the parent node whenever we branch (which we do when the optimal solution has one or more variables with fractional value that should be integer).

From Figure 3.12 we see that every point in the feasible region for the integer problem is included in the feasible region for either subproblem 2 or 3. Moreover, these two subproblems have no point in common. We have obtained this partition by branching *on variable x_1.*

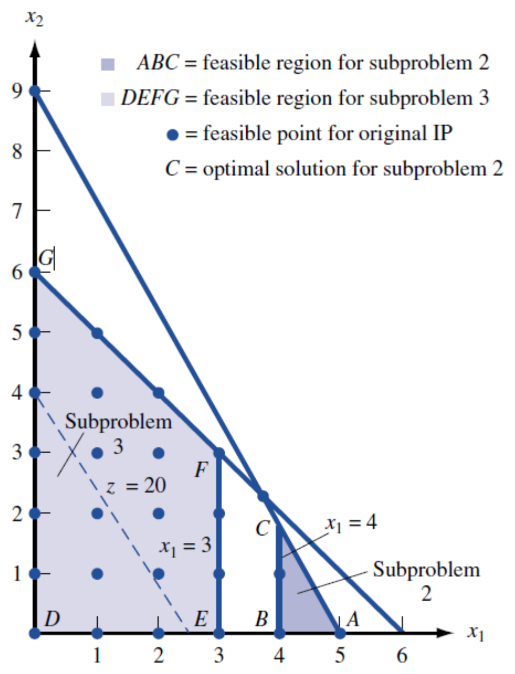


Figure 3.12: Feasible Region for Subproblems 2 and 3.

We now choose any subproblem that has not yet been solved as a linear problem. We arbitrarily choose to solve subproblem 2 first. From Figure 3.12 we see that the optimal solution to subproblem 2 is $\tilde{x} = \left(4, \frac{9}{5}\right)$ *(point C) with optimal value* $z = 41$*. The accomplishments to date are summarized in Figure 3.13. With the label* t *we are indicating the chronological order in which the subproblems are solved, but it is not necessary to include this in the tree if we do not wish so.*

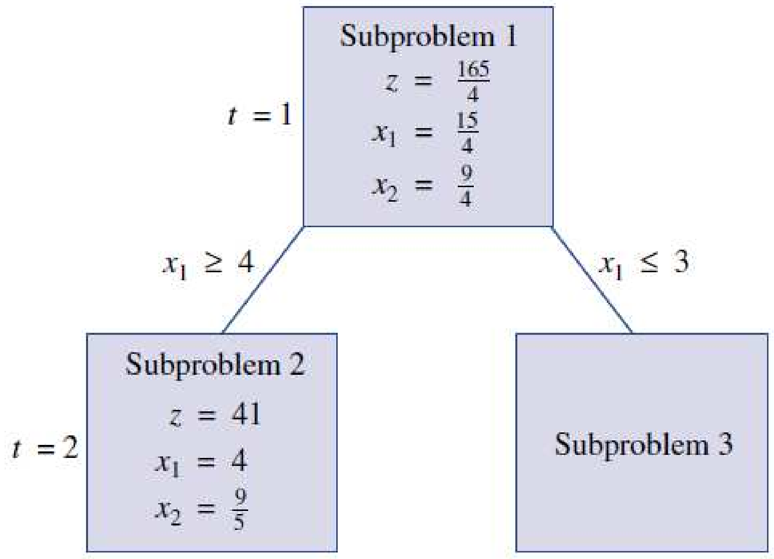


Figure 3.13: Subproblems 2 and 3 Solved.

The optimal solution to subproblem 2 did not yield an integer solution. So, we choose to use subproblem 2 to create two new subproblems. We choose a fractional-valued variable in the optimal solution to subproblem 2 and then branch on that variable. Because x_2 is the only fractional variable in this case the choice is simple. We branch on x_2 by creating subproblems that include constraints $x_2 \geq 2$ and $x_2 \leq 1$. This creates the following two subproblems:

- *Subproblem 4: Subproblem 1 + constraints $x_1 \geq 4$ and $x_2 \geq 2$ = subproblem 2 + constraint $x_2 \geq 2$.*
- *Subproblem 5: Subproblem 1 + constraints $x_1 \geq 4$ and $x_2 \leq 1$ = subproblem 2 + constraint $x_2 \leq 1$.*

The feasible regions for subproblems 4 and 5 are displayed in Figure 3.14. The set of unsolved subproblems consists of subproblems 3,4, and 5.

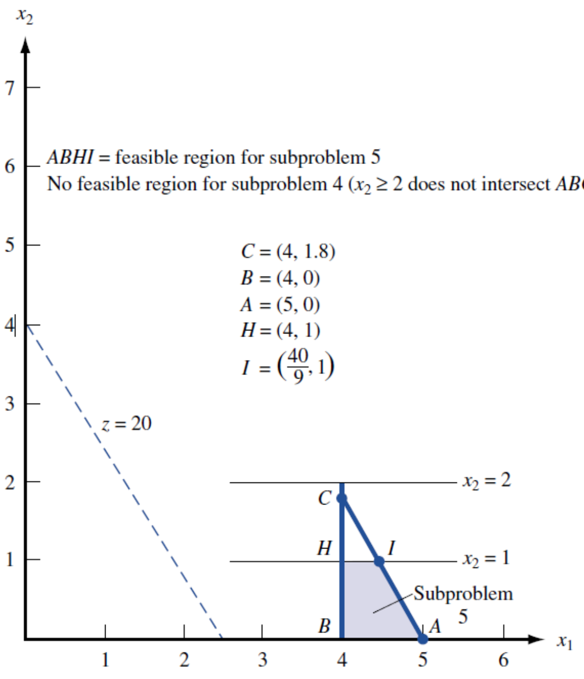


Figure 3.14: Feasible Region for Subproblems 4 and 5.

We now choose a subproblem to solve. In this case, we choose the most recently created subproblem (why we do this is not discussed here). There are two subproblems that tie for this criterion: subproblems 4 and 5. We arbitrarily choose to solve subproblem 4.

*From Figure 3.14 we see that subproblem 4 is infeasible: there are no feasible points. Thus, this subproblem cannot yield an optimal solution to our integer problem and it is discarded (*pruned *or* fathomed*). We have indicated this with a cross on the branching tree (see Figure 3.15) and we do not need to continue exploring this subproblem any more.*

Now, the only unsolved subproblems are subproblems 3 and 5. Following the same criterion as before (solve the most recently created subproblem), we are going to solve subproblem 5 next. From Figure 3.14 we see that the optimal solution to subproblem 5 is point I: $\tilde{x} = \left(\frac{40}{9}, 1\right)$ and $z = \frac{365}{9}$. This solution does not yield any immediately useful information. So, we partition its feasible region by branching on x_1, which creates two new subproblems:

- *Subproblem 6: subproblem 5 + constraint $x_1 \geq 5$.*

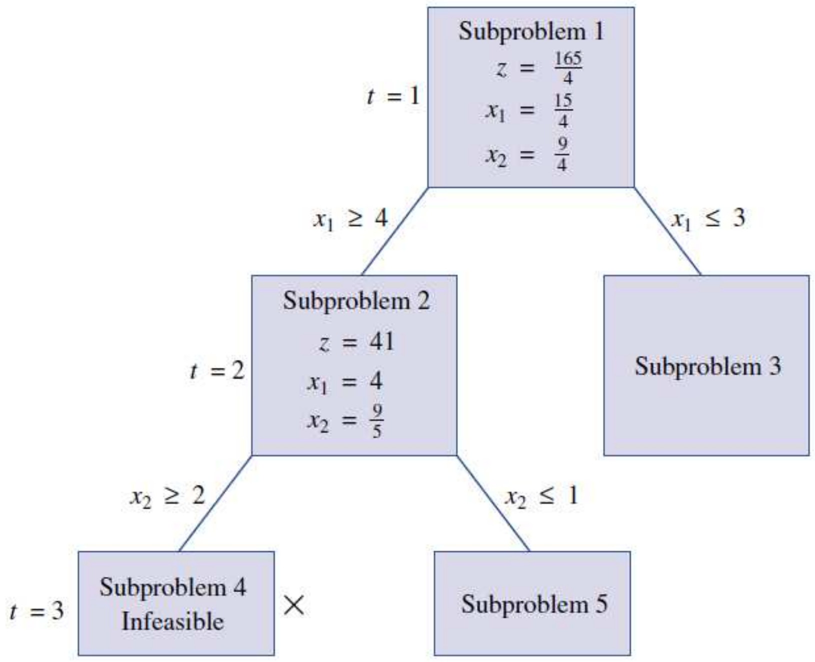


Figure 3.15: Subproblems 1,2, and 4 Solved.

- *Subproblem 7: subproblem 6 + constraint $x_1 \leq 4$.*

Our branching tree looks as show in Figure 3.17.

Subproblems 3,6, and 7 are now unsolved. The most recently created subproblems are 6 and 7. We arbitrarily choose to solve subproblem 7 first.

From Figure 4.1 we see that the optimal solution to subproblem 7 is point H: $\tilde{x} = (4, 1)$ and $z = 37$. Both components are integer. So, this solution is feasible for the original problem. Thus, further branching from this subproblem is not required because we will not find a better solution and we prune this node. See Figure 3.18.

*Every time that we find and integer solution, we update our best integer solution so far (*incumbent solution*). In this case, this is the first that we find, so, it is saved as the incumbent, $\bar{x}$. The* incumbent value *is the objective value of this solution and it is represented with $\bar{z}$. We now know that the optimal value is at least 37. We need to explore the rest of the tree to see if a better solution exists. The value of 37 constitutes a lower bound to the optimal value.*

The only remaining unsolved subproblems are 6 and 3. Following the usual rule, we next solve subproblem 6. From Figure 4.1 we find that the optimal solution to this subproblem is $\tilde{x} = (5, 0)$ and $z = 40$. This is an integer solution that improves the incumbent solution. Therefore:

- *We update the incumbent solution information: $\bar{x} = (5, 0)$, $\bar{z} = 40$.*
- *We prune the node associated to subproblem 6 because we cannot find a better solution here.*

The information is summarized in Figure 3.20.

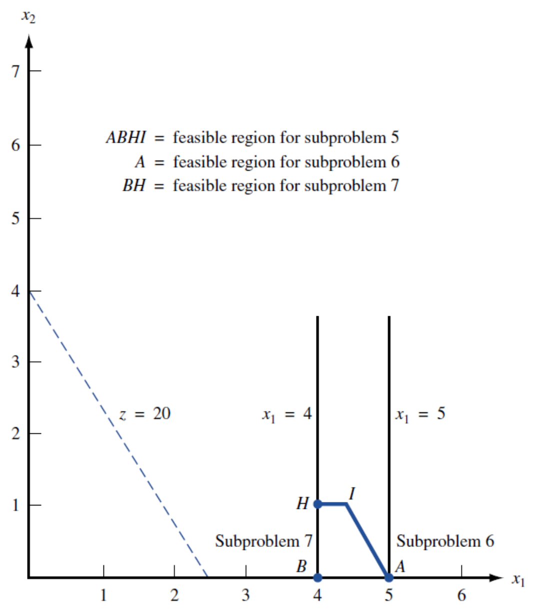


Figure 3.16: Feasible Region for Subproblems 6 and 7.

Now subproblem 3 is the only unsolved subproblem. From Figure 3.12 we find that the optimal solution to this subproblem is $\tilde{x} = (3, 3)$ *and* $z = 39$*. It is an integer solution that does not improve the current solution. So, we prune this node and there are no more subproblems to solve. We can conclude that the optimal solution is* $x^* = (5, 0)$ *and the optimal value is* $z^* = 40$*. In terms of the problem, the optimal policy is to manufacture 5 tables and no chair. The profit is £40.*

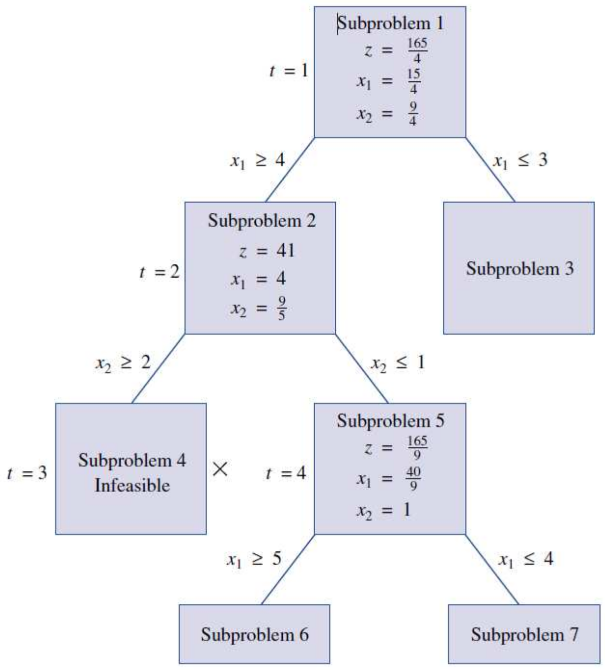


Figure 3.17: Subproblems 1,2,4, and 5.

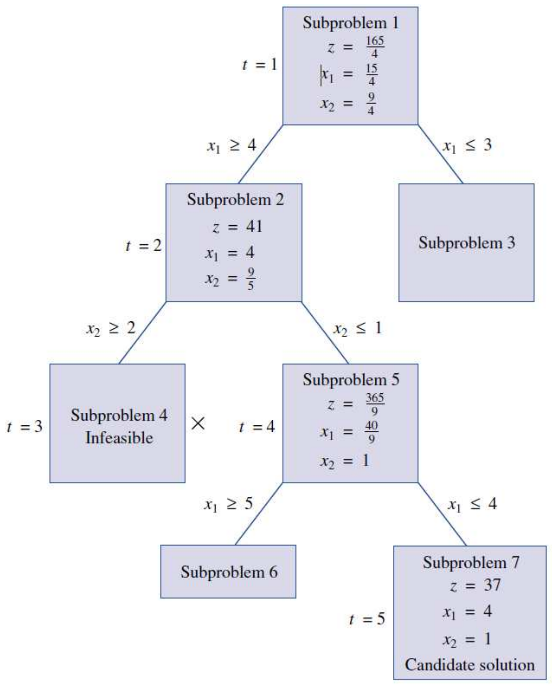


Figure 3.18: Branch-and-Bound Tree After Five Subproblems Have Been Solved.

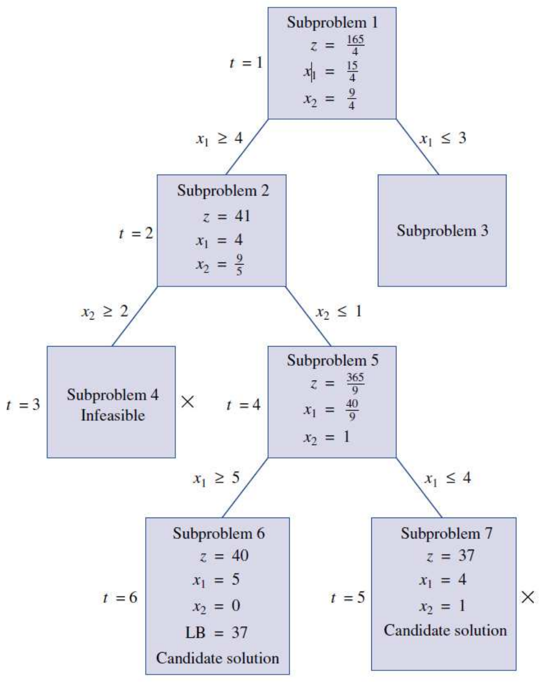


Figure 3.19: Branch-and-Bound Tree After Six Subproblems Have Been Solved.

Branch-and-Bound for Mixed Integer Programming Problems

We have seen how to use branch-and-bound for pure integer programming problems. If we have a problem where some of the variables are not required to be integers, then we modify

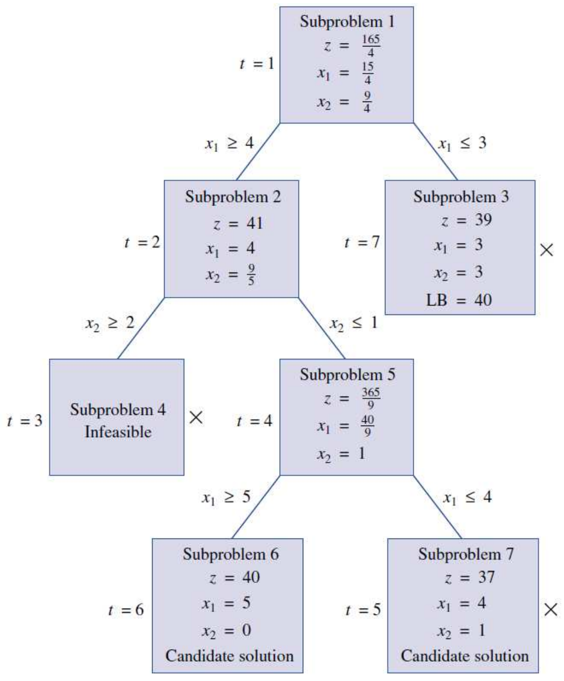


Figure 3.20: Final Branch-and-Bound Tree.

the previous method by branching only on variables that are required to be integers. Also, for a solution to a subproblem to be a candidate solution, it needs only that those variables required to be integer take integer values.

Let us see an example.

Example 3.13
Let us solve the following mixed integer problem with branch-and-bound:

$$\begin{array}{ll} Max. & 2x_1 + x_2 \\ s.t. & 5x_1 + 2x_2 \leq 8, \\ & x_1 + x_2 \leq 3, \\ & x_1, x_2 \geq 0, \\ & x_1 \text{ integer.} \end{array}$$

Note: The lecture notes are not showing the graphical resolution of the subproblems for this example.

As before, we begin by solving the linear relaxation of the integer problem. The optimal solution is $\tilde{x} = \left(\frac{2}{3}, \frac{7}{3}\right)$ and the optimal value is $\frac{11}{3}$. Both variables have fractional values.

Because x_2 is allowed to be fractional, we do not need to branch on this variable. Thus, we branch on x_1. This yields Subproblems 2 (when we add constraint $x_1 \leq 0$) and 3 (when we add constraint $x_1 \geq 1$).

We choose to solve next Subproblem 2. The optimal solutionis $\tilde{x} = (0, 3)$ and the optimal value is 3. As x_1 is integer, this is a feasible solution for the original integer problem. Besides, it is the first feasible solution that we obtain. Thus, it becomes the incumbent solution: $\bar{x} = (0, 3)$ and $\bar{z} = 3$.

We solve now subproblem 3. Its optimal solution is $\tilde{x} = \left(1, \frac{3}{2}\right)$, with optimal value $\frac{7}{2}$. As x_1 is integer, this is another candidate solution. We prune the current node and we compare this solution with our incumbent solution. In this case, this solution is better because it has a higher objective value and we are maximizing. Thus, it becomes our new incumbent solution. Moreover, as there are no more nodes to explore, this is our optimal solution: $x^* = \left(1, \frac{3}{2}\right)$, with optimal value $z^* \frac{7}{2}$.

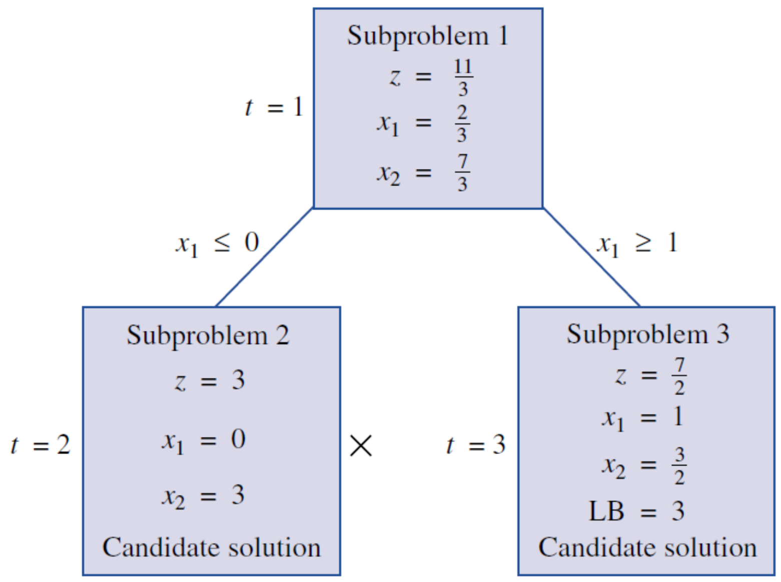


Figure 3.21: Branch-and-Bound for a Mixed Integer Problem.

3.9. Cuts for Integer Programming Problems

In the previous example we have seen that, when we solve some of the subproblems when using the branch-and-bound method, we obtain fractional solutions. If we look at the feasible region of these continuous linear optimization problems, we see that there are many more fractional points than integer points (actually, there is an uncountable amount of fractional points). If our feasible region (for the linear relaxation) were smaller, then we would have more chances of finding an integer point as the optimal solution to the linear relaxation and, hence, the algorithm would end sooner because we would solve faster all the nodes that need to be solved.

In order to have a smaller fractional feasible region without losing any feasible integer point, we add **cuts**: inequalities that are satisfied by all the feasible integer points but that some of the fractional points do not satisfy, and, thus, are cut off. For example, if in the previous example we consider exclusively Subproblem 2 (let us forget that this a subproblem of the full problem, or that Subproblem 3 exists), then $x_2 \leq 1$ is a cut.

Unfortunately, in general it is very difficult to find cuts. It is highly problem dependent. This said, there are some techniques that can be used to obtain some general cuts. Here we will see one of them to obtain the so called **Gomory cuts**. But first we are going to see a useful technique that we will need later to derive the Gomory cuts.

3.9.1. Coefficient Splitting

For constraints with non-integer coefficient cuts can be derived by splitting off the fractional part of each coefficient.

We need that all the variables are on the left-hand side of the constraint.

Constraints with $\leq$

For a "$\leq$" constraint we need to split off the positive fractional part of each coefficient.

Example 3.14
Assume $x_1, x_2, x_3 \geq 0$ and integer and consider that we are solving a problem where one of the constraints is

$$2x_1 + \frac{6}{5}x_2 - \frac{3}{5}x_3 \leq \frac{19}{5}.$$

For every variable that is integer and whose coefficient in the inequality is fractional, we split this coefficient into integer and positive fractional part:

$$2x_1 + \left(1 + \frac{1}{5}\right)x_2 + \left(-1 + \frac{2}{5}\right)x_3 \leq \frac{19}{5}.$$

Next we move the terms with fractional coefficient to the right hand side:

$$2x_1 + x_2 - x_3 \leq \frac{19}{5} - \frac{1}{5}x_2 - \frac{2}{5}x_3.$$

Now, observe that, as $x_2, x_3 \geq 0$, we have that $\frac{19}{5} - \frac{1}{5}x_2 - \frac{2}{5}x_3 \leq \frac{19}{5}$. Therefore, it holds that

$$2x_1 + x_2 - x_3 \leq \frac{19}{5}.$$

The next remark is that the left-hand side of the previous inequality is an integer number because all the variables involved and their coefficients are integer. Therefore, if we represent with $\lfloor x \rfloor$ the integer part of a number x, it holds that

$$2x_1 + x_2 - x_3 \leq \left\lfloor \frac{19}{5} \right\rfloor = 3.$$

Constraints with $\geq$

For a "$\geq$" constraint we need to split off negative fractional parts of each coefficient (or we multiply the inequality by -1 and we apply what we have learned for $\leq$ inequalities).

Assume $x_1, x_2, x_3 \geq 0$ and integer and consider that we are solving a problem where one of the constraints is

$$2x_1 + \frac{6}{5}x_2 - \frac{3}{5}x_3 \geq \frac{19}{5}.$$

We proceed as before, but now we split off the negative fractional part of each coefficient.

$$2x_1 + \left(2 - \frac{4}{5}\right)x_2 - \frac{3}{5}x_3 \geq \frac{19}{5};$$

$$2x_1 + 2x_2 \geq \frac{19}{5} + \frac{4}{5}x_2 + \frac{3}{5}x_3 \geq \frac{19}{5}.$$

Now, if we represent by $\lceil x \rceil$ the smallest integer number greater than or equal to x, it holds that

$$2x_1 + 2x_2 \geq \left\lceil \frac{19}{5} \right\rceil = 4.$$

If we divide by 2, then

$$x_1 + x_2 \geq 2.$$

Unfortunately, there is no way of know how useful this technique is before applying it. Sometimes we will obtain inequalities that reduce the fractional feasible region of the problem. But some other times, the inequality will not help at all. There is, though, a case where we can guarantee that we can obtain an inequality that contributes to reducing the fractional feasible region: the Gomory cuts that we will study next.

3.9.2. Gomory Cuts

When we are using branch-and-bound to solve an integer problem and we solve one of the (continuous) linear problems that appear as part of this process, if we obtain a solution that is fractional (that is, not all the integer variables take integer values), then we have seen that we can create two new subproblems by adding the inequalities that we have seen in the branch-and-bound method. Then we solve those subproblems and check again. However, there is an alternative: we could try to add an inequality that cuts off the fractional optimal solution obtained when solving the continuous problem and solve that new problem, whose optimal solution will be different. There are different methods to add such cuts. One of them is to use Gomory cuts, which we are going to see here. Let us learn the method through an example.

Example 3.15
Assume we have solved the linear relaxation of the following integer problem:

$$\begin{array}{ll} Max. & -x_1 + 2x_2 \\ s.t. & 2x_1 + 2x_2 \geq 3, \\ & -x_1 + x_2 \leq 1, \\ & 2x_1 + x_2 \leq 9, \\ & x_1, x_2 \in \mathbb{Z}^+. \end{array}$$

The optimal solution is $\tilde{x} = \left(\frac{8}{3}, \frac{11}{3}\right)$. It is on the intersection of inequalities $-x_1 + x_2 \leq 1$ and $2x_1 + x_2 \leq 9$.

Now, we rewrite these constraints as equalities by introducing slack variables $s_3, s_4 \geq 0$:

$$-x_1 + x_2 + s_3 = 1 \qquad (3.10)$$

$$2x_1 + x_2 + s_4 = 9 \qquad (3.11)$$

Note that since x_1 and x_2 are integer and the right hand-sides are integer, then s_3 and s_4 must be integer as well.

Solving this system for x_1 and x_2 gives

$$x_1 = \tfrac{8}{3} + \tfrac{1}{3}s_3 - \tfrac{1}{3}s_4 \qquad (3.12)$$

$$x_2 = \tfrac{11}{3} - \tfrac{2}{3}s_3 - \tfrac{1}{3}s_4 \qquad (3.13)$$

An equality can be seen as simultaneous "$\leq$" and "$\geq$" constraints. We can therefore apply the coefficient splitting method for both cases to either of (3.12) and (3.13). This allows to derive up to four new cuts.

For example, if we take (3.12), we have the equality

$$x_1 - \frac{1}{3}s_3 + \frac{1}{3}s_4 = \frac{8}{3}. \qquad (3.14)$$

If see it as inequality less than or equal to, then we have the following:

$$x_1 - \frac{1}{3}s_3 + \frac{1}{3}s_4 \leq \frac{8}{3};$$

$$x_1 + \left(-1 + \frac{2}{3}\right)s_3 + \frac{1}{3}s_4 \leq \frac{8}{3};$$

$$x_1 - s_3 \leq \frac{8}{3} - \frac{2}{3}s_3 - \frac{1}{3}s_4 \leq \frac{8}{3};$$

$$x_1 - s_3 \leq \left\lfloor \frac{8}{3} \right\rfloor = 2.$$

Now, we can write this inequality in the original variables by using expression (3.10) *to substitute:*

$$x_1 - 1 - x_1 + x_2 \leq 2;$$

$$x_2 \leq 3$$

.

As can be seen, point $\tilde{x}$ is cut off.

If, instead we see (3.14) *as a greater than or equal to inequality, then we have the following:*

$$x_1 - \frac{1}{3}s_3 + \frac{1}{3}s_4 \geq \frac{8}{3};$$

$$x_1 - \frac{1}{3}s_3 + \left(1 - \frac{2}{3}\right)s_4 \geq \frac{8}{3};$$

$$x_1 + s_4 \geq \frac{8}{3} + \frac{1}{3}s_3 + \frac{2}{3}s_4 \geq \frac{8}{3};$$

$$x_1 + s_4 \geq \left\lceil \frac{8}{3} \right\rceil = 3.$$

Now, we use (3.13) *to write this inequality in the original variables:*

$$x_1 + 9 - 2x_1 - x_2 \geq 3;$$

$$-x_1 - x_2 \geq -6;$$

$$x_1 + x_2 \leq 6.$$

Chapter 4

Game Theory

Many situations in life include conflict and competition. Think of military battles, political campaigns, marketing campaigns, etc. A basic feature is that the final outcome depends upon the combination of strategies selected by all the adversaries involved. Game Theory deals with the general features of competitive situations like these in a formal and abstract way. There is a strong emphasis on the decision-making processes of adversaries.

We will focus first on the simplest case: **two-person zero-sum games**. As the name implies, these games involve only two adversaries, usually called **players**. They are called zero-sum games because one player wins whatever the other one loses, so that the sum of their total net winnings is zero.

4.1. Formulation of Two-Person Zero-Sum Games

Let us consider a game called odds and evens. Each of the two players shows simultaneously either one or two coins. If the number of coins matches, so that the total number is even, then player 1 wins £1 from player 2. If the number does not much, so that the total number is odd, then player 1 pays £1 to player 2.

Each player has two possible **strategies**: to show either one coin or two coins. The resulting payoff to player 1 in pounds is shown in the **payoff table** given in Table 4.1:

Strategy		Player 2	
		1	2
Player 1	1	1	-1
	2	-1	1

Table 4.1: Payoff Table for the Odds and Evens Game.

In general, a two-person game is characterized by:

1. The strategies of player 1.
2. The strategies of player 2.

3. The payoff table.

Before the game begins, each player knows the strategies that he has available, the ones that the opponent has available, and the payoff table. The play of the game consists of each player simultaneously choosing a strategy without knowing the opponent's choice.

The payoff table shows the gain (positive or negative) for player 1 that would result from each combination of strategies for the two players. It is given only for player 1 because, as this is a zero-sum game, the payoff table for player 2 is obtained just by multiplying this one by -1.

The entries in the payoff table represent the utility to player 1 of each outcome.

A main objective of Game Theory is the development of rational criteria for selecting a strategy. Two fundamental assumptions are:

1. Both players are rational.
2. Both players choose strategies to promote their own welfare.

4.2. Basic Concepts

We are going to see some basic concepts of Game Theory through an example.

Example 4.1 (The Two Politicians)
Two Politicians are running against each other for an election. Campaign plans must be made for the final two days, which will be spent campaigning in two key cities: Edinburgh and Glasgow. To avoid wasting campaign time, they plan to travel at night, and spend either one full day in each city or two full days in just one of the cities. However, since the necessary arrangements need to be made in advance, neither politician will learn his opponent's campaign schedule until after he has finished his own. Therefore, each politician has asked his campaign manager in each of these cities to assess what the impact would be (in terms of votes won or lost) from the various possible combinations of days spent there by himself and by his opponent. He then wishes to use this information to choose his best strategy on how to use these two days.

4.2.1. Formulation as a Two-Person Zero-Sum Game

To formulate this problem as a two-person zero-sum game, we must identify the two players (which is easy: the two politicians), the strategies for each player, and the payoff table.

Each player has the following three strategies:

1. Strategy 1: One day in each city.
2. Strategy 2: Two days in Edinburgh.
3. Strategy 3: Two days in Glasgow.

Each entry in the payoff table for player 1 represents the utility to player 1 (or the negative utility to player 2) of the outcome resulting from the corresponding strategies used by the two players. From the player's viewpoints, the objective is to win votes. Therefore, the appropriate entries for the payoff table for player 1 are the total net votes won from the opponent. Using units of thousands of votes, the payoff table is shown in Table 4.2. For the moment, only the form of the table is shown, with the numbers missing.

Strategy		Player 2		
		1	2	3
	1			
Player 1	2			
	3			

Table 4.2: Payoff Table for Politician 1.

4.2.2. Variation 1

Let us assume that the payoff table for player 1 is given by Table 4.3. The goal is to determine which strategy should each player select.

Strategy		Player 2		
		1	2	3
	1	1	2	4
Player 1	2	1	0	5
	3	0	1	-1

Table 4.3: Payoff Table for Politician 1 (Variation 1).

We are going to see that this is a special case that can be solved by applying the concept of dominated strategies to rule other strategies that will always be inferior. A strategy is **dominated** by another strategy if the second strategy is always at least as good (and maybe sometimes better) regardless of what the opponent does. A dominated strategy can be eliminated from the set of strategies.

In Table 4.3, strategy 3 for player 1 is dominated by strategy 1 because the latter has larger payoffs: $(1, 2, 4) \geq (0, 1, -1)$. Looking at this inequality, it is a component-wise comparison of vectors: $1 \geq 0$, $2 \geq 1$, and $4 \geq -1$.

When we eliminate strategy 3, the table looks like this:

	1	2	3
1	1	2	4
2	1	0	5

As both players are assumed to be rational, player 2 knows that player 1 has only two possible strategies. Therefore, now player 2 has a dominated strategy: strategy 3, which is dominated by both strategies 1 and 2. We reach this conclusion for player 2 by remembering that his payoff table is the negative of the payoff table of player 1. Thus, he is interested in having values as small as possible in the payoff table of player 1 because the goal is to have as many votes as possible. In our case, $(1, 1) \leq (4, 5)$. So, we delete strategy 3 for player 2 and the remaining table looks like this:

	1	2
1	1	2
2	1	0

Next, it is easy to see that strategy 2 for player 1 is dominated by strategy 1: $(1, 2) \geq (1, 0)$. Thus, strategy 2 is deleted:

	1	2
1	1	2

Finally, strategy 2 for player 2 is dominated by strategy 1 because $1 < 2$. We eliminate strategy 2:

	1
1	1

Therefore, both players should select strategy 1. Player 1 will win 1,000 votes from player 2.

The **value of the game** is the payoff to player 1 when both players play optimally. A **fair game** is a game whose value is 0. Thus, our example is not a fair game.

In this case, we have been able to solve the game by using exclusively the analysis of dominated strategies. However, that is not usually the case, as we will see next.

4.2.3. Variant 2

Suppose that now the payoff table for player 1 is given by Table 4.4. In this case we cannot solve the problem by dominated strategies. Thus, it is not that obvious what to do.

Strategy		Player 2		
		1	2	3
	1	-3	-2	6
Player 1	2	2	0	2
	3	5	-2	-4

Table 4.4: Payoff Table for Politician 1 (Variation 2).

Let us consider player 1. By selecting strategy 1, he could win 6 or he could lose as much as -3. But remember that player 2 is rational. If he knows that player 1 is playing strategy 1, he will try to minimize his loss. Thus, he will play strategy 1, which will turn out in player 1 losing 3. But player 1 is also rational. If he knows that player 2 is playing strategy 1, he will try to maximize his payoff. Thus he will play strategy 3. And so on so forth.

It seems then reasonable if players play a strategy where they look for a best guarantee. If player 1 players strategy 1, he could lose up to 3. If he plays strategy 2, his minimum gain will be 0. And if he plays strategy 3, then he could lose up to 6. Therefore, strategy 2 seems to be a “rational” choice against his rational opponent, player 2. No matter what player 2 does, player 1 will have a minimum payoff of 0. It is a line of reasoning that assumes that both players are averse to risking larger losses than necessary, contrary to gambling for large payoffs with chances of losing larger amounts.

Consider now player 2. He can lose as much as 5, with strategy 1, or 6, with strategy 3. However, with strategy 2 he will lose nothing, and he could even win 2. By the same logic than before, it seems that his best strategy against a rational opponent is strategy 2.

If both players choose strategy 2, both have a payoff of 0. Moreover, neither player improves upon his best guarantee, but both are forcing the opponent into the same position. Even when the opponent deduces a player's strategy, the opponent cannot exploit this information to improve his position.

In this line of reasoning each player should play in such a way as to minimize his maximum loss whenever the resulting choice of strategy cannot be exploited by the opponent to then improve his position. This is the **minimax criterion**: select a strategy that would be best even if the selection were being announced to the opponent before the opponent chooses a strategy. In terms of the payoff table, it implies that player 1 should select the strategy whose minimum payoff is maximum, whereas player 2 should choose the strategy whose maximum payoff is minimum. In the example of Table 4.4, strategy 2 is the maximin strategy for player 1, and strategy 2 is the minimax strategy for player 2. The resulting payoff of 0 is the value of the game. So, it is a fair game.

It must be notice that the same entry in this payoff table yields both the maximin and the minimax values. The position of any such entry is called a **saddle point**.

Because of the saddle point, neither opponent can take advantage of the opponent's strategy to improve his own position. Thus,neither player has any motive to consider changing strategies, either to take advantage of this opponent, or to prevent the opponent from taking advantage of him. This is a **stable solution**, also called a **equilibrium position**.

4.2.4. Variation 3

Assume now that the payoff tables is Table 4.5, and suppose that both players attempt to apply the minimax criterion in the same way as before.

Strategy		Player 2		
		1	2	3
Player 1	1	0	-2	2
	2	5	4	-3
	3	2	3	-4

Table 4.5: Payoff Table for Politician 1 (Variation 3).

Player 1 can guarantee that he will lose no more than 2 by playing strategy 1. Similarly, player 2 can guarantee that he will lose no more than 2 by playing strategy 3. However, the maximin value is -2 and the minimax value is 2. They do not coincide. Thus, there is no saddle point. Let us see the implications.

Player 1 would win 2 from player 2, but this would make player 2 unhappy because his minmax strategy has a payoff of 2. Because player 2 is rational and can anticipate this outcome, he would then conclude that he can do much better by playing strategy 2, which has a payoff of -2 for player 1. Now, because player 1 is rational, he would anticipate this switch and conclude that he can improve from -2 to 4 by changing to strategy 2. Realizing this, player 2 would then switch back to strategy 3 to convert a loss of 4 into a gain of 3. this would cause again player 1 to go back to strategy 1, and we would enter into a loop. Therefore, even if the game is played only once, any tentative choice of a strategy leaves that player with a motive to consider changing strategies.

The suggested solution, player 1 to play strategy 1 and player 2 to play strategy 3, is an **unstable solution**. The payoff table does not have any saddle point. So, how should players choose their strategies here? The key is not to be predictable because, whenever a player knows the strategy of the opponent, he can take advantage of this knowledge. thus, it seems better to choose randomly among a set of acceptable strategies. But how?

4.3. Games with Mixed Strategies

Whenever a game does not possess a saddle point, game theory advises each player to assign a probability distribution over his set of strategies. Let:

- x_i be the probability that player 1 will use strategy i, $i = 1, 2, \ldots, m$,
- y_j be the probability that player 2 will use strategy j, $j = 1, 2, \ldots, n$,

where m and n are the respective numbers of available strategies. Thus, player 1 would specify his plan for playing the game by assigning values to $x_1, x_2, \ldots, x_m$. Because these values are probabilities, they are nonnegative and add to 1. Likewise for player 2 and $y_1, y_2, \ldots, y_n$.

These plans $(x_1, x_2, \ldots, x_m)$ and $(y_1, y_2, \ldots, y_n)$ are usually referred to as **mixed strategies**, and the original strategies are called **pure strategies**.

When the game is played, each player uses one of the pure strategies. But which one is chosen randomly according to the probability distribution specified by the mixed strategy. for example, if $(x_1, x_2, x_3) = (0.5, 0.5, 0)$, then player 1 chooses between the first two strategies with the same probability and does not play strategy 3.

There is no completely satisfactory measure of performance for evaluating mixed strategies. But one that is useful is the **expected payoff**. By application the definition of expected value, this quantity is

$$\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij} x_i y_j,$$

for player 1, where p_{ij} is the payoff for player 1 if player 1 plays pure strategy i and player 2 plays pure strategy j. This value indicates what the average payoff will tend to if the game is played many times.

By using this measure we can extend the concept of the minimax criterion to games that lack a saddle point, and, thus, need mixed strategies. In this context the minimax criterion says that a given player should select the mixed strategy that minimizes the maximum expected loss to himself. Equivalently, when the focus is on payoffs (player 1) rather than losses (player 2), this criterion becomes maximin instead, that is, maximize the minimum expected payoff to the player. This optimal value for player 1 is represented with $\underline{v}$: maximum minimum expected payoff. The optimal value for player 2 is represented with $\overline{v}$: minimum maximum expected loss.

For a game with only pure strategies, we may have that $\underline{v} \neq \overline{v}$, as we have seen in Variant 3 of the example. For a stable game, we have that $\underline{v} = \overline{v}$. Particularly, the following result tells us that this always happens for mixed strategies:

Theorem 4.2 (Minimax Theorem)
If mixed strategies are allowed, the pair of mixed strategies that is optimal according to the minimax criterion provides a stable solution with $\underline{v} = \overline{v} = v$ (the value of the game), so that neither player can do better by unilaterally changing his strategy.

Now we need to see how we can determine what the mixed strategies should be to achieve a stable solution.

4.4. Graphical Solution Procedure

Consider any game with mixed strategies such that, after dominated strategies are eliminated, one of the two players has only two pure strategies. Let us assume that it is player 1. Because of the mixed strategies (x_1, x_2), we know that $x_2 = 1 - x_1$. Taking this into account, we plot the expected payoff as a function of x_1 for each of the pure strategies of player 2. Then we will use this graph to find a point that maximizes the minimum expected payoff.

Let us consider Variation 3 of the example that we have studied (Table 4.5). After eliminating the third pure strategy for player 1, which is dominated by strategy 2, the payoff table is reduced to Table 4.6.

Strategy		Player 2		
		1	2	3
	1	0	-2	2
Player 1	2	5	4	-3

Table 4.6: Payoff Table for Politician 1 (Variation 3) without Strategy 3 for Player 1.

For each of the pure strategies of player 2, the expected payoff for player 1 is shown in the following table:

(y_1, y_2, y_3)	Expected Payoff for Player 1
$(1, 0, 0)$	$0 \cdot x_1 + 5(1 - x_1) = 5 - 5x_1$
$(0, 1, 0)$	$-2 \cdot x_1 + 4(1 - x_1) = 4 - 6x_1$
$(0, 0, 1)$	$2 \cdot x_1 - 3(1 - x_1) = -3 + 5x_1$

Now, we plot these expected payoff lines on a graph, as shown in Figure **??**. For any given values of x_1 and (y_1, y_2, y_3), the expected payoff will be the appropriate weighted average of the corresponding points on these three lines. That is, the expected payoff for player 1 is

$$y_1(5 - 5x_1) + y_2(4 - 6x_1) + y_3(-3 + 5x_1).$$

Remember that player 2 wants to minimize this expected payoff for player 1. Given x_1, player 2 can minimize this expected payoff by choosing the pure strategy that corresponds to the bottom line for that x_1. Note that, in this particular case, it will never be line $5 - 5x_1$ because there is always another line below for any x_1.

According to the minimax criterion (which is a maximin criterion from the viewpoint of player 1), player 1 wants to maximize this minimum expected payoff. Consequently, player 1 should select the value of x_1 where the bottom line peaks. In this case, this is where lines $-3 + 5x_1$ and $4 - 6x_1$ intersect, which yields an expected payoff of

$$\underline{v} = v = \max_{0 \le x_1 \le 1} \{\min\{-3 + 5x_1, 4 - 6x_1\}\}.$$

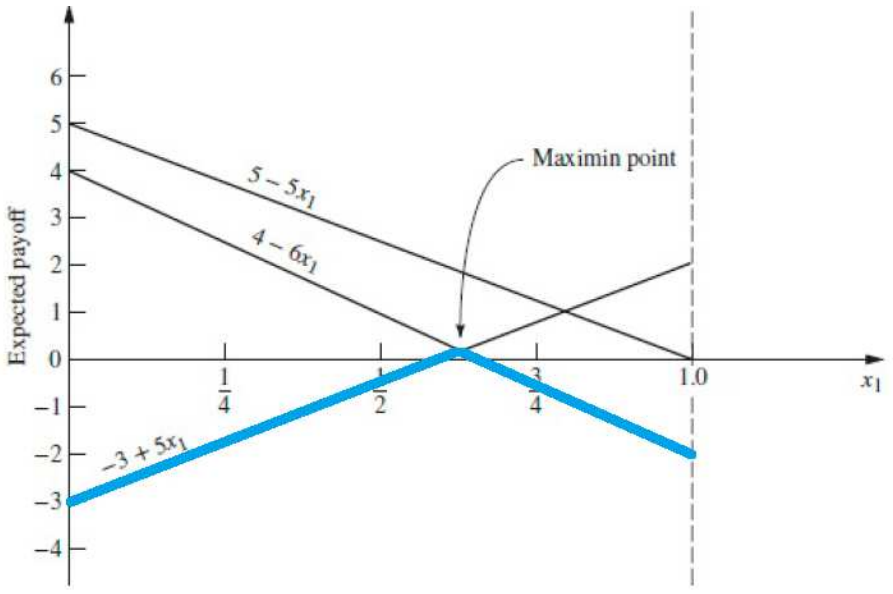


Figure 4.1: Expected Payoff Lines.

As we have said that eh value of x_1 that we are looking for is the intersection of the two mentioned lines, we solve $-3+5x_1 = 4-6x_1$ to obtain that $x_1 = \frac{7}{11}$.

Therefore, the optimal mixed strategy for player 1 is $(x_1^*, x_2^*) = \left(\frac{7}{11}, \frac{4}{11}\right)$ and the value of the game is $v = -3+5\cdot\frac{7}{11} = \frac{2}{11}$.

To compute optimal values for (y_1, y_2, y_3), we remember that for $x_1 = \frac{7}{11}$ the expected payoff for player 1 is $v = \frac{2}{11}$. When we substitute in the expected payoff expression, we have that

$$\frac{20}{11}y_1 + \frac{2}{11}y_2 + \frac{2}{11}y_3 = \frac{2}{11}.$$

That is, $10y_1 + y_2 + y_3 = 1$.

In addition, we know that $y_1 + y_2 + y_3 = 1$. So, it follows that $y_1^* = 0$.

we have then that only strategies 2 and 3 need to be considered. This is again the previous situation where there are only two nondominated pure strategies. We repeat the same kind of arguments, but now for player 2. That is, $y_3 = 1 - y_2$ and the expected payoffs for player 1 are given by:

(x_1, x_2)	Expected Payoff for Player 1
$(1,0)$	$-2\cdot y_2 + 2(1-y_2) = 2-4y_2$
$(0,1)$	$4\cdot y_2 - 3(1-y_2) = -3+7y_2$

We are interested in the point where $2-4y_2$ and $-3+7y_2$ cut. Thus, $y_2^* = \frac{5}{11}$ and the optimal mixed strategy for player 2 is $(y_1^*, y_2^*, y_3^*) = \left(0, \frac{5}{11}, \frac{6}{11}\right)$.

4.5. Solution with Linear Programming

When we have a game that uses mixed strategies, we can use linear programming to find what these strategies should be.

Remember that the expected payoff for player 1 is

$$\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_i y_j,$$

where p_{ij} is the payoff for player 1 if strategy i is played by player 1 and strategy j is played by player 2.

It can be shown (we do not prove it here) that the optimal mixed strategies $(x_1, x_2, \ldots, x_m)$ and $(y_1, y_2, \ldots, y_n)$ can be found by solving the following two problems:

$$\begin{array}{ll}
\text{Max.} & x_{m+1} \\
\text{s.t.} & p_{11}x_1 + p_{21}x_2 + \ldots + p_{m1}x_m - x_{m+1} \geq 0, \\
 & p_{12}x_1 + p_{22}x_2 + \ldots + p_{m2}x_m - x_{m+1} \geq 0, \\
 & \vdots \\
 & p_{1n}x_1 + p_{2n}x_2 + \ldots + p_{mn}x_m - x_{m+1} \geq 0, \\
 & x_1 + x_2 + \ldots + x_m = 1, \\
 & x_1, x_2, \ldots, x_m \geq 0.
\end{array}$$

$$\begin{array}{ll}
\text{Min.} & y_{n+1} \\
\text{s.t.} & p_{11}y_1 + p_{12}y_2 + \ldots + p_{1n}y_n - y_{n+1} \leq 0, \\
 & p_{21}y_1 + p_{22}y_2 + \ldots + p_{2n}y_n - y_{n+1} \leq 0, \\
 & \vdots \\
 & p_{m1}y_1 + p_{m2}y_2 + \ldots + p_{mn}y_n - y_{n+1} \leq 0, \\
 & y_1 + y_2 + \ldots + y_n = 1, \\
 & y_1, y_2, \ldots, y_n \geq 0.
\end{array}$$

Here x_{m+1} and y_{n+1} are auxiliary variables. They are not restricted, that is, they are allowed to take negative values. But they take a positive value in an optimal solution (we will not prove this).

Example 4.3
Let us solve Variation 3 using this method.

In order to find the mixed strategies (x_1, x_2), we need to solve the following problem:

$$\begin{array}{ll}
\textit{Max.} & x_3 \\
\textit{s.t.} & 5x_2 - x_3 \geq 0, \\
 & -2x_1 + 4x_2 - x_3 \geq 0, \\
 & 2x_1 - 3x_2 - x_3 \geq 0, \\
 & x_1 + x_2 = 1, \\
 & x_1, x_2 \geq 0.
\end{array}$$

This problem has 3 variables. Thus, we cannot solve it graphically. By using an optimization solver (or the simplex method by hand) we obtain that the optimal solution is $x^* = \left(\frac{7}{11}, \frac{4}{11}, \frac{2}{11}\right)$.

The first two values of the solution are the mixed strategy. Obviously, they coincide with the values that we obtained when we solved this game with the graphical method. The last value, x_3^*, is not just some random value. It is precisely the value of the game.

In a similar way, in order to find the mixed strategies (y_1, y_2, y_3), *we solve the following problem:*

$$\begin{array}{ll} \textit{Min.} & y_4 \\ \textit{s.t.} & -2y_2 + 2y_3 - y_4 \leq 0, \\ & 5y_1 + 4y_2 - 3y_3 - y_4 \leq 0, \\ & y_1 + y_2 + y_3 = 1, \\ & y_1, y_2, y_3 \geq 0. \end{array}$$

Again, we cannot solve this problem graphically. But using a suitable method we obtain the solution $y^* = \left(0, \frac{5}{11}, \frac{6}{11}, \frac{2}{11}\right)$.

As before, the first three values give the mixed strategy, and the last value is the value of the game. Particularly, observe how the optimal value of the two auxiliary variables is the same.