MA2601 - Operational Research

Handbook

Dr Geraint Palmer-Liyu

Spring Semester 2026

Contents

Introduction

- Dr Geraint Palmer-Liyu
- Abacws M/2.54
- palmergi1@cardiff.ac.uk

Operational research involves using an analytical approach to help better decision making. This course is in two halves, modelling and optimisation.

Modelling

Modelling here involves developing a mathematical representation of a system to help with understanding and prediction.

We will consider four topics:

1. **Probability Distributions** - Find probabilities and summary statistics of Poisson and Exponential random variables. Deriving expressions for their the PDFs, CDFs, and summary statistics.

2. **Markov Chains** - Modelling scenarios as a set of discrete states, with probabilistic transitions between states. We will consider both discrete-time and continuous-time Markov chains.

3. **Queueing Theory** - Studying models of queues, where customers arrive randomly, wait in a line until a server is free, spend some time in service, and then leave. We will derive queueing formulas using Markov chains, and consider some cost analysis.

4. **Stochastic Simulation** - Understanding stochastic systems by using a computer to randomly generate random events and their interactions. We will first consider Monte Carlo simulation to generate random events, and then discrete event simulation of queueing systems.

Optimisation

Optimisation here involves maximising or minimising some function subject to limited resources which are expressed as constraints. Optimisation is everywhere! For example minimising production costs, maximising the value of some advertising campaign, minimising the distance of a route, maximising profit, and so on.

We will consider four topics:

5. **Linear Programming** - Formulating and solving real-life problems which can be expressed in terms of a linear objective function that is to be maximised or minimised, and a set of linear constraints.
6. **Transportation Problems** - Formulating and solving the specific problem of transporting items from supplier to consumer at minimum cost. Other real-life problems can be modelled as a transportation problem and solved in this way.
7. **Dynamic Programming** - The problem of optimising routes through a network, and formulating seemingly unrelated problems as problems on graphs.
8. **Project Management** - The problem of determining the length of a project and which tasks are critical to it finishing on time. Subsequently we will look at how we calculate the optimal project plan given that we can reduce the duration of some tasks, albeit at a price.

Chapter 1

Probability Distributions

Learning Outcomes:

- Be able to use and understand the Poisson distribution;
- Be able to derive summary statistics for the Poisson distribution;
- Be able to use and understand and Exponential distribution;
- Be able to derive summary statistics for the Exponential distribution.

1.1 Introduction

A *random variable* the outcome of a random experiment. More formally, it is a function that maps the outcome of that random experiment to a real number. This can be discrete or continuous. A probability distribution describe how a random variable behaves. More formally it is a function that gives the probability of occurrence of the outcome of a random experiment. Recall that there are two types of distributions:

- *probability density functions*, PDFs for continuous random variables
  (or *probability mass functions*, PMFs for discrete random variables)
- *cumulative distribution functions*, CDFs.

PDFs and PMFs will usually be denoted by f, while CDFs are denoted by F. Recall also that for continuous random variables, $f = F'$.

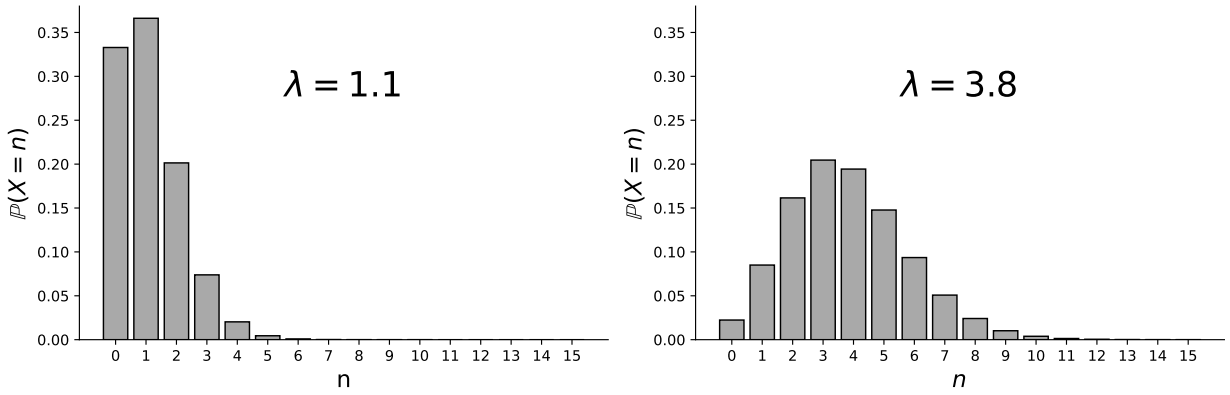
One particularly useful discrete probability distribution for this course is the Poisson distribution, and a related continuous probability distribution, the Exponential distribution.

1.2 The Poisson Distribution

Consider a series of events that occur at random, at a mean rate of λ per time unit. Consider that each event is independent of the others, that is that the occurrence of one event does not affect the occurrence of another. Let X be a random variable representing the number of these events in some time interval. Then X follows a Poisson distribution with PMF:

$$\mathbb{P}(X = n) = \frac{\lambda^n e^{-\lambda}}{n!} \tag{1.1}$$

An visual example of this PMF for $\lambda = 1.1$ and $\lambda = 3.8$:



The PMF of Equation 1.1 can be derived from the assumptions about the random variable. This is given below, for interest:

Poisson Derivation

First consider one small time interval δt, which is so small we assume that the probability of more than one event occurring is $\mathcal{O}(\delta t)$. Let $\mathbb{P}(n, I)$ denote the probability of there being n events in an interval of length I. By definition we have:

- $\mathbb{P}(1, \delta t) = \lambda \delta t + \mathcal{O}(\delta t)$
- $\mathbb{P}(0, \delta t) = 1 - \lambda \delta t + \mathcal{O}(\delta t)$
- $\mathbb{P}(n, \delta t) = \mathcal{O}(\delta t)$ if $n > 1$

And from now on we can ignore when $n > 1$ and the $\mathcal{O}(\delta t)$ terms. Using these facts we will prove by induction that $\mathbb{P}(X = n) = \frac{\lambda^n e^{-\lambda}}{n!}$ for all $n \in \mathbb{N}$.

The Base Case

First we will show that $\mathbb{P}(X = 0) = e^{-\lambda t}$: Consider the interval $(0, t + \delta t)$. Note that the intervals $(0, t)$ and $(t, t + \delta t)$ are independent, and it doesn't matter where we start the interval, it is only the interval length which is of interest. Therefore:

$$\begin{aligned}\mathbb{P}(0, t+\delta t) &= \mathbb{P}(0,t)\mathbb{P}(0,\delta t) \\ &= \mathbb{P}(0,t)(1-\lambda\delta t)\end{aligned}$$

rearranging we get:

$$\frac{\mathbb{P}(0,t+\delta t)-\mathbb{P}(0,t)}{\delta t} = -\lambda\mathbb{P}(0,t)$$

and taking the limit as $\delta t \to 0$:

$$\frac{d\mathbb{P}(0,t)}{dt} = -\lambda\mathbb{P}(0,t)$$

which is a differential equation. With initial condition $\mathbb{P}(0,0) = 0$, which must be true, this has solution:

$$\mathbb{P}(0,t) = e^{-\lambda t}$$

The Inductive Step

We now assume that $\mathbb{P}(X=n) = \frac{\lambda^n e^{-\lambda}}{n!}$ for all $n \leq k$. How about $\mathbb{P}(X = k+1)$?
Again consider the interval $t+\delta t$, and again note that the intervals of length t and δt are independent. Now we have:

$$\begin{aligned}\mathbb{P}(n+1, t+\delta t) &= \mathbb{P}(n+1,t)\mathbb{P}(0,\delta t) + \mathbb{P}(n,t)\mathbb{P}(1,\delta t) \\ &= \mathbb{P}(n+1,t)(1-\lambda\delta t) + \mathbb{P}(n,t)\lambda\delta t\end{aligned}$$

rearranging we get:

$$\frac{\mathbb{P}(n+1,t+\delta t)-\mathbb{P}(n+1,t)}{\delta t} + \lambda\mathbb{P}(n+1,t) = \lambda\mathbb{P}(n,t)$$

and taking the limit as $\delta t \to 0$:

$$\begin{aligned}\frac{d\mathbb{P}(n+1,t)}{dt} + \lambda\mathbb{P}(n+1,t) &= \lambda\mathbb{P}(n,t) \\ &= \frac{\lambda^{n+1}e^{-\lambda}}{n!}\end{aligned}$$

this is a first order differential equation. Solving, with the initial condition $\mathbb{P}(n+1,0) = 0$, gives:

$$\mathbb{P}(n+1,t) = \frac{\lambda^{n+1}e^{-\lambda}}{(n+1)!}$$

which completes the derivation.

The Poisson distribution has equal mean and variance. Let $X \sim \text{Poisson}(\lambda)$, then:

$$\mathbb{E}[X] = \lambda \tag{1.2}$$
$$\text{Var}(X) = \lambda \tag{1.3}$$

Proofs:

Poisson Mean

Theorem: Let $X \sim \text{Poisson}(\lambda)$. Then $\mathbb{E}[X] = \lambda$.
Proof:

$$\mathbb{E}[X] = \sum_{n=0}^{\infty} n\mathbb{P}(X = n)$$

We can ignore the first term as it is equal to zero:

$$= \sum_{n=1}^{\infty} n\mathbb{P}(X = n)$$

Substitute in the PMF expression:

$$= \sum_{n=1}^{\infty} n\frac{\lambda^n e^{-\lambda}}{n!}$$

Factor scalar multiple out of the sum:

$$= e^{-\lambda} \sum_{n=1}^{\infty} \frac{n}{n!}\lambda^n$$

Simplify the fraction:

$$= e^{-\lambda} \sum_{n=1}^{\infty} \frac{1}{(n-1)!}\lambda^n$$

Factor scalar multiple out of the sum:

$$= \lambda e^{-\lambda} \sum_{n=1}^{\infty} \frac{1}{(n-1)!}\lambda^{n-1}$$

Recognise the exponential expansion:

$$= \lambda e^{-\lambda} e^{\lambda}$$

Simplify:

$$= \lambda$$

Poisson Variance

Theorem: Let $X \sim \text{Poisson}(\lambda)$. Then $\text{Var}(X) = \lambda$.
Proof: A similar argument to above yields $\mathbb{E}[X^2] = \lambda(\lambda + 1)$ (check this!). Now

$$\begin{aligned}\text{Var}(X) &= \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \\ &= \lambda(\lambda + 1) - \lambda^2 \\ &= \lambda^2 + \lambda - \lambda^2 \\ &= \lambda\end{aligned}$$

We can now answer some questions about Poisson distributed variables.

Example 1 *At a bus stop, bus arrivals are known to be Poisson distributed with rate 4 per hour.*
a) What is the probability, after arriving at the bus stop, that I need to wait more than 20 minutes for a bus?
b) If I sit for 30 minutes, what is the probability that more than two buses will have arrived?

Solution to Example 1 *In turn:*
a) Let X represent the number of buses arriving in 20 minutes. Then $X \sim$ Poisson $\left(\frac{4}{3}\right)$. Then:

$$\begin{aligned}\mathbb{P}(\textit{no buses in 20 minutes}) &= \mathbb{P}(X = 0) \\ &= \frac{\left(\frac{4}{3}\right)^0 e^{-\frac{4}{3}}}{0!} \\ &= e^{-\frac{4}{3}} = 0.2636\end{aligned}$$

b) Let Y represent the number of buses arriving in 30 minutes. Then $Y \sim$ Poisson(2). Then:

$$\begin{aligned}\mathbb{P}(\textit{more then 2 buses in 30 minutes}) &= \mathbb{P}(X > 2) \\ &= 1 - \mathbb{P}(X \leq 2) \\ &= 1 - \mathbb{P}(X = 0) - \mathbb{P}(X = 1) - \mathbb{P}(X = 2) \\ &= 1 - \frac{2^0 e^{-2}}{0!} - \frac{2^1 e^{-2}}{1!} - \frac{2^2 e^{-2}}{2!} \\ &= 1 - 0.1353 - 0.2707 - 0.2707 \\ &= 0.3233\end{aligned}$$

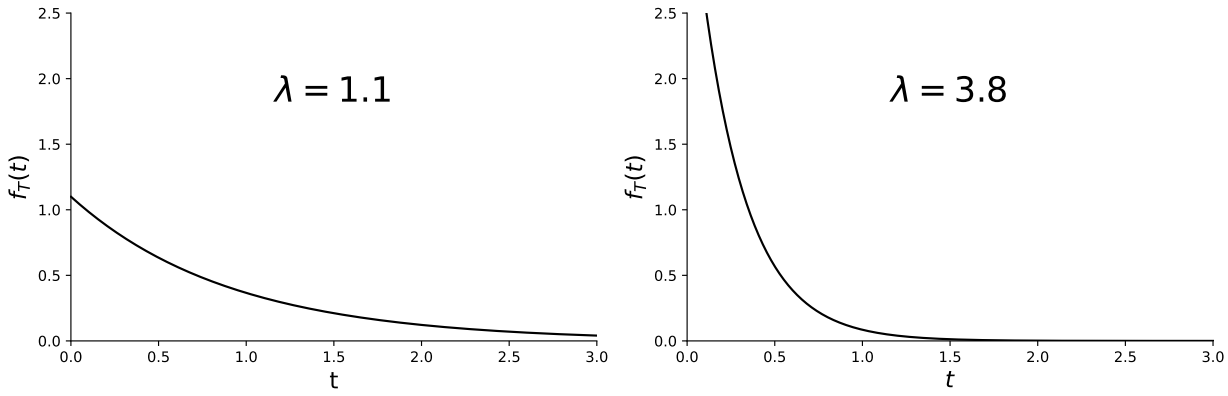
1.3 The Exponential Distribution

The Poisson distribution describes the number of random events in a given time unit. The Exponential distribution describes the time between random events. Let T be a random variable representing the time between two consecutive Poisson events that occur at a rate of λ per time unit. Then T follows an Exponential distribution with PDF and CDF:

$$f_T(t) = \mathbb{P}(T = t) = \lambda e^{-\lambda t} \tag{1.4}$$

$$F_T(t) = \mathbb{P}(T < t) = 1 - e^{-\lambda t} \tag{1.5}$$

An visual example of this PDF for $\lambda = 1.1$ and $\lambda = 3.8$:



The CDF and PDF can be derived from the Poisson distribution:

Exponential Derivation

Let T be the random variable representing the time between two consecutive Poisson events that occur at a rate of λ per time unit, and let X be the random variable representing the number of events that occur in a time interval of length t. Then $X \sim \text{Poisson}(\lambda t)$.

Now consider the CDF of T:

$$\begin{aligned} F_T(t) = \mathbb{P}(T < t) &= 1 - \mathbb{P}(T \geq t) \\ &= 1 - \mathbb{P}(X = 0) \\ &= 1 - \frac{(\lambda t)^0 e^{-\lambda t}}{0!} \\ &= 1 - e^{-\lambda t} \end{aligned}$$

For the PDF:

$$\begin{aligned} f_T(t) &= F_T'(t) \\ &= \frac{d}{dt}\left[1 - e^{-\lambda t}\right] \\ &= \lambda e^{-\lambda t} \end{aligned}$$

The Exponential distribution has mean and variance related to the inverse of its rate. Let $T \sim \text{Expon}(\lambda)$, then:

$$\mathbb{E}[T] = \frac{1}{\lambda} \tag{1.6}$$

$$\text{Var}(T) = \frac{1}{\lambda^2} \tag{1.7}$$

Proofs:

Exponential Mean

Theorem: Let $T \sim \text{Expon}(\lambda)$. Then $\mathbb{E}[T] = \frac{1}{\lambda}$.
Proof:

$$\begin{aligned} \mathbb{E}[T] = \int_0^\infty t f_t(t) dt &= \lambda \int_0^\infty t e^{-\lambda t} dt \\ &= \lambda \left[-\frac{1}{\lambda} t e^{-\lambda t} + \int \frac{1}{\lambda} e^{-\lambda t} \right]_0^\infty \\ &= \left[-t e^{-\lambda t} - \frac{1}{\lambda} e^{-\lambda t} \right]_0^\infty \\ &= 0 - 0 - 0 + \frac{1}{\lambda} = \frac{1}{\lambda} \end{aligned}$$

Exponential Variance

Theorem: Let $T \sim \text{Expon}(\lambda)$. Then $\text{Var}(T) = \frac{1}{\lambda^2}$.
Proof: A similar argument to above yields $\mathbb{E}[T^2] = \frac{2}{\lambda^2}$ (check this using integration by parts twice!). Now

$$\begin{aligned}\text{Var}(T) &= \mathbb{E}[T^2] - (\mathbb{E}[X])^2 \\ &= \frac{2}{\lambda^2} - \frac{1}{\lambda^2} = \frac{1}{\lambda^2}\end{aligned}$$

Memorylessness

An important and surprising property of the Exponential distribution is that it is *memoryless*. This means that the probability of waiting an extra t amount of time for an event, given that we have already been waiting a amount of time, is the same as just waiting t amount of time. That is:

$$\mathbb{P}(T > t + a \mid T > a) = \mathbb{P}(T > t) \tag{1.8}$$

That is to say, it does not matter how long you have already waited for an event, the probability of it happening within the next t time units is the same regardless.

Memorylessness

Theorem: The Exponential distribution is memoryless. That is, let $T \sim \text{Expon}(\lambda)$. Then $\mathbb{P}(T > t + a \mid T > a) = \mathbb{P}(T > t)$.
Proof:

From the definition of conditional probability:

$$\mathbb{P}(T > t + a \mid T > a) = \frac{\mathbb{P}(T > t + a \cap T > a)}{\mathbb{P}(T > a)}$$

As the second event in the numerator is contained withing the first:

$$= \frac{\mathbb{P}(T > t + a)}{\mathbb{P}(T > a)}$$

Substituting in the CDF expressions:

$$\begin{aligned}&= \frac{1 - F_T(t + a)}{1 - F_T(a)} \\ &= \frac{e^{-\lambda(t+a)}}{e^{-\lambda a}}\end{aligned}$$

And simplifying:

$$\begin{aligned} &= e^{-\lambda(t)} \\ &= \mathbb{P}(T > t) \end{aligned}$$

We can now answer some questions about Exponentially distributed variables.

Example 2 *The time until a lithium battery runs out of energy can be modelled as an Exponential random variable with rate* 0.3 *per week.*

a) What is the expected lifetime of a lithium battery?

b) I know that I have been using a battery for 2 weeks. What is the probability that it will run out of energy in the next week?

c) I buy a pack of 6 batteries. What is the probability that they will all run out of energy within 3 weeks?

Solution to Example 2 *Let* T *represent the lifetime of a lithium battery. Then* $T \sim$ *Expon*(0.3) *when our time units are in weeks.*

a) The average lifetime of a battery is $\mathbb{E}[T] = \frac{1}{\lambda} = \frac{1}{0.3} = 3.333$ *weeks.*

b) We make use of the memoryless property:

$$\begin{aligned} \mathbb{P}(T < 3 \mid T > 2) &= \mathbb{P}(T < 1) \\ &= 1 - e^{-\lambda 1} \\ &= 1 - e^{-0.3} = 0.259 \end{aligned}$$

c) Let $T_i \sim$ *Expon*(0.3) *represent the lifetime of battery* i*. Now as all* T_i *are independent and identically distributed:*

$$\begin{aligned} \mathbb{P}\left(\bigcap_{i \leq 6} (T_i < 3)\right) &= (\mathbb{P}(T < 3))^6 \\ &= \left(1 - e^{\lambda t}\right)^6 \\ &= \left(1 - e^{0.3 \times 3}\right)^6 = 0.044 \end{aligned}$$

1.4 Thinning & Superposition

We have seen that a Poisson process relates to two different random variables, a Poisson random variable that counts the number of random events in a time interval; and an Exponential random variable that describes the times between consecutive events. Poisson processes can be combined and decomposed, this is called superposition and thinning.

Poisson Superposition

Let $Y_1, Y_2, \ldots, Y_n$ be independent Poisson random variables with rates $\lambda_1, \lambda_2, \ldots, \lambda_n$. Let $Y = \sum_i^n Y_n$. Then:

$$Y \sim \text{Poisson}\left(\sum_i^n \lambda_i\right) \tag{1.9}$$

Analogously, let $X_1, X_2, \ldots, X_n$ be independent Exponentially distributed random variables with rates $\lambda_1, \lambda_2, \ldots, \lambda_n$. Let $X = \min_i X_n$. Then:

$$X \sim \text{Expon}\left(\sum_i^n \lambda_i\right) \tag{1.10}$$

Poisson Thinning

Consider a Poisson process with rate λ, letting $Y \sim \text{Poisson}(\lambda)$ be the random variable representing the number of such events per time unit. Suppose that each event can be labelled as one of N possible labels, and that the probability of an event being of label i is p_i. Now consider Y_i, the number of events with label i in a time unit. Then:

$$Y_i \sim \text{Poisson}(p_i\lambda) \tag{1.11}$$

Analogously, let X be an Exponentially distributed random variable representing the time between two consecutive events in the Poisson process. Then $X \sim \text{Expon}(\lambda)$. Let X_i represent the times between events with label i. Then:

$$X_i \sim \text{Expon}(p_i\lambda) \tag{1.12}$$

Proofs can be derived by considering the CDFs in each case.

1.5 The Erlang Distribution

Another related distribution is the Erlang distribution, with two parameters, the rate parameter λ, and shape parameter k, the sum of independent Exponential random variables:

Erlang Distribution

Let $X_i \sim \text{Expon}(\lambda)$ independent random variables, then:

$$Y = \sum_{i=1}^{k} X_i \sim \text{Erlang}(k, \lambda) \tag{1.13}$$

and:

$$f_Y(t) = \frac{\lambda^k t^{k-1} e^{-\lambda t}}{(k-1)!} \tag{1.14}$$

The PDF is given without derivation. However, due to the fact that Y is the sum of k independent Exponential random variables, its mean and variance can be found:

Erlang Mean

Theorem: Let $Y \sim \text{Erlang}(k, \lambda)$. Then $\mathbb{E}[Y] = \frac{k}{\lambda}$.
Proof: Using the fact that for any two independent random variables A and B we have $\mathbb{E}[A + B] = \mathbb{E}[A] + \mathbb{E}[B]$, then the result follows:

$$\mathbb{E}[Y] = \sum_{i=1}^{k} \mathbb{E}[X_i] = \sum_{i=1}^{k} \frac{1}{\lambda} = \frac{k}{\lambda}$$

Erlang Variance

Theorem: Let $Y \sim \text{Erlang}(k, \lambda)$. Then $\text{Var}(Y) = \frac{k}{\lambda^2}$.
Proof: Using the fact that for any two independent random variables A and B we have $\text{Var}(A + B) = \text{Var}(A) + \text{Var}(B)$, then the result follows:

$$\text{Var}(Y) = \sum_{i=1}^{k} \text{Var}(X_i) = \sum_{i=1}^{k} \frac{1}{\lambda^2} = \frac{k}{\lambda^2}$$

Unlike the Exponential distribution, the Erlang distribution is not memoryless.

Chapter 2

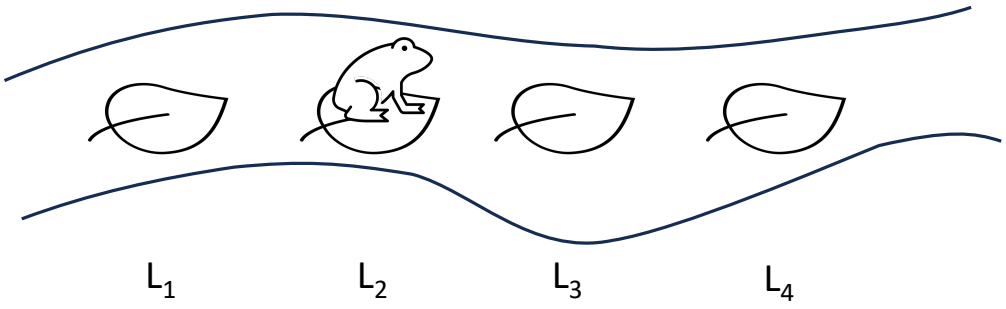
Markov Chains

Learning Outcomes:

- Be able to model with discrete-time Markov chains;
- Be able to model with continuous-time Markov chains;
- Be able to classify states of a Markov chain.

2.1 Introduction

It is often possible to describe the behaviour of some stochastic system by describing all the different states that it could be in, and how the system moves between these states. Let's motivate this with an example - consider a river with four lily-pads, and a frog jumping from one to the other randomly every time unit, as shown in the image below. We'll assume that the frog will only jump to an adjacent lily-pad, and will choose between lily-pads with equal probability.



We can describe the system as being in one of four states: either the frog is on the first lily-pad (L_1); the frog is on the second lily-pad (L_2); the frog is on the third lily-pad (L_3); or the frog is on the fourth lily-pad (L_4). This assumes that the time the frog is jumping through the air is instantaneous. The set $S = \{L_1, L_2, L_3, L_4\}$ is called the *state space* of the system.

We also know something about the probability of transitioning from one state to another each time step. Let p_{ij} represent the probability of reaching state j if you are currently in state i, then the following *transition probability matrix* represents all the probabilities of transitioning between all states:

$$P = \begin{array}{c} \\ L_1 \\ L_2 \\ L_3 \\ L_4 \end{array} \begin{array}{c} \begin{array}{cccc} L_1 & L_2 & L_3 & L_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

That is p_{L_2,L_3}, denoting that the probability of the frog being on the third lily-pad, given that it is currently on the second lily-pad, is $1/2$.

One realisation of this might be:

Time	1	2	3	4	5	6	7	8	9	10
State	L_1	L_2	L_1	L_2	L_3	L_4	L_3	L_2	L_3	L_2

This is an example of a *stochastic process*. In particular, this is an example of a particular type of stochastic process called a discrete-time Markov chain.

2.2 Discrete-Time Markov Chains

Discrete-time Markov chains are a particular type of discrete-time stochastic process.

Discrete-Time Stochastic Process

A discrete-time stochastic process is a random vector, whose components are indexed by time. It can also be thought of as a family of random variables:

$$\{X_n,\ n \in \mathbb{N}\}$$

A discrete-time Markov chain is a discrete-time stochastic process, where the outcome of each X_n is an element of a countable set of states, called the *state space*, and that satisfies the *Markov property*:

Markov Property (Discrete-Time)

The Markov property states that X_{n+1} depends only on X_n, and not on any previous states. That is:

$$\mathbb{P}(X_{n+1} = x_{n+1} \mid X_n = x_n, X_{n-1} = x_{n-1}, \dots X_0 = x_0) = \mathbb{P}(X_{n+1} = x_{n+1} \mid X_n = x_n)$$

In this way, we can define our frog leaping example as a Markov chain. We need only define the *state space* and *transition probability matrix*. Note that the state space may be infinite.

- The state space S is a set of states. It is useful to order the states $S = \{s_0, s_1, \dots\}$.
- A valid transition probability matrix, P, has entries $p_{ij} = \mathbb{P}(X_n = s_j \,|X_{n-1} = s_i\,)$, where $0 \leq p_{ij} \leq 1$ which are valid probabilities, and rows that sum to 1.

With these tools we can analyse the system. For example, if the frog is currently on the second lily-pad, that is the system is in state L_2:

- *what is the probability that they will be on the third lily-pad (L_3) in one time step?*

  We can read this directly from the transition probability matrix:

$$\mathbb{P}(X_1 = L_3 \mid X_0 = L_2) = 1/2$$

- *what is the probability that they will be on the third lily-pad (L_3) in two time steps?*

  We need to consider the probability of being in particular states in X_1. There are two states that could reach L_3: either L_2 or L_4:

$$\begin{aligned}\mathbb{P}(X_2 = L_3 \mid X_0 = L_2) &= \mathbb{P}(X_2 = L_3 \mid X_1 = L_2)\mathbb{P}(X_1 = L_2 \mid X_0 = L_2)\\ &\quad + \mathbb{P}(X_2 = L_3 \mid X_1 = L_4)\mathbb{P}(X_1 = L_4 \mid X_0 = L_2)\\ &= (1/2 \times 0) + (1/2 \times 0) = 0\end{aligned}$$

- *what is the probability that they will be on the third lily-pad (L_3) in three time steps?*

  we now have to consider all possibly routes to L_3 in three time steps:

  - $L_2 \to L_1 \to L_2 \to L_3$
  - $L_2 \to L_3 \to L_2 \to L_3$
  - $L_2 \to L_3 \to L_4 \to L_3$

  therefore:

$$\begin{aligned}\mathbb{P}(X_3 = L_3 \mid X_0 = L_2) &= \mathbb{P}(X_3 = L_3 \mid X_2 = L_2)\mathbb{P}(X_2 = L_2 \mid X_1 = L_1)\mathbb{P}(X_1 = L_1 \mid X_0 = L_2)\\ &\quad + \mathbb{P}(X_3 = L_3 \mid X_2 = L_2)\mathbb{P}(X_2 = L_2 \mid X_1 = L_3)\mathbb{P}(X_1 = L_3 \mid X_0 = L_2)\\ &\quad + \mathbb{P}(X_3 = L_3 \mid X_2 = L_4)\mathbb{P}(X_2 = L_4 \mid X_1 = L_3)\mathbb{P}(X_1 = L_3 \mid X_0 = L_2)\\ &= (1/2 \times 1 \times 1/2) + (1/2 \times 1/2 \times 1/2) + (1 \times 1/2 \times 1/2)\\ &= 5/8\end{aligned}$$

The calculations done here are called the Chapman-Kolmogorov equations, and are equivalent to matrix multiplication. Let $\underline{\pi}_n$ be a vector representing the probabilities of being in each state at time n, and the transition probability matrix is denoted by P, then:

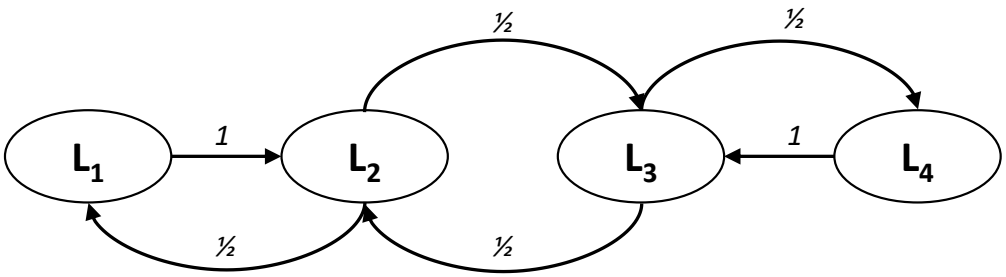
DTMC Transitive States

$$\underline{\pi}_{n+1} = \underline{\pi}_n P$$

and by repeated application of this equation:

$$\underline{\pi}_{n+1} = \underline{\pi}_0 P^n$$

It is often useful to visualise Markov chains. A common way to do this is to represent states in ovals, and transitions as arrows between states, labelled by the probabilities. For example, for the frog-leaping example:

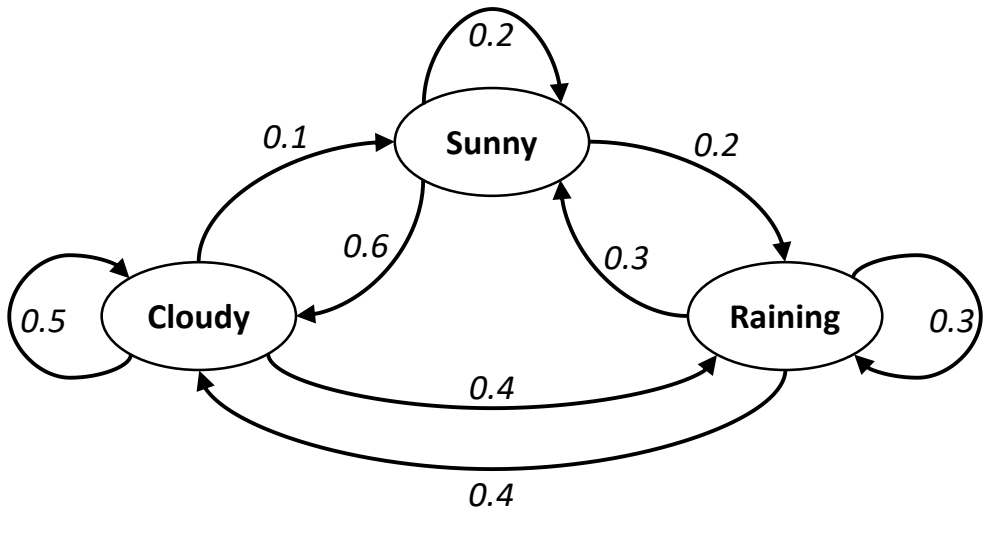


Example 3 *The weather in Cardiff can be either Sunny, Cloudy or Raining:*

- *When Sunny, the probability of it remaining sunny the next day is 0.2, of it clouding over is 0.6, and of it raining is 0.2;*
- *When Cloudy, the probability of it remaining cloudy the next day is 0.5, the probability of the sun coming out is 0.1, and the probability of it raining is 0.4;*
- *When Raining, the probability of it continuing to rain the next day is 0.3, of the sun coming out is 0.3, and of it being cloudy is 0.4.*

Draw the Markov chain and give the transition probability matrix.
Mari is getting married in Cardiff in three days time. The weatherman says that tomorrow there will be 50% chance of sun, 50% chance of cloud, and 0% chance of it raining. What is the probability that it will rain on Mari's wedding day?

Solution to Example 3 *Drawing the Markov chain gives:*



The transition probability matrix, ordering the states Sunny, Cloudy, Raining, is:

$$P = \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.1 & 0.5 & 0.4 \\ 0.3 & 0.4 & 0.3 \end{pmatrix}$$

The weatherman has stated that $\underline{\pi}_1 = (0.5, 0.5, 0.0)$. *Mari is getting married at* $n = 3$, *therefore:*

$$\begin{aligned}
\underline{\pi}_3 &= \underline{\pi}_1 P^2 \\
&= (0.5, 0.5, 0.0) \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.1 & 0.5 & 0.4 \\ 0.3 & 0.4 & 0.3 \end{pmatrix}^2 \\
&= (0.5, 0.5, 0.0) \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.1 & 0.5 & 0.4 \\ 0.3 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.1 & 0.5 & 0.4 \\ 0.3 & 0.4 & 0.3 \end{pmatrix} \\
&= (0.5, 0.5, 0.0) \begin{pmatrix} 0.16 & 0.5 & 0.34 \\ 0.19 & 0.47 & 0.34 \\ 0.19 & 0.5 & 0.31 \end{pmatrix} \\
&= (0.175, 0.485, 0.34)
\end{aligned}$$

Therefore there is a 34% chance of raining on Mari's wedding day.

Steady-State Analysis

We have seen that the rows of P^n gives the probabilities of being in each state in n time steps, given the probability of currently being in each state. It can be seen experimentally that for many[1] transition probability matrices, if we take n to be very large, then we end up with a matrix where each row is identical.

For example, consider the weather Markov chain from Example 3:

$$
\begin{aligned}
P &= \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.1 & 0.5 & 0.4 \\ 0.3 & 0.4 & 0.3 \end{pmatrix} \\
P^2 &= \begin{pmatrix} 0.16 & 0.5 & 0.34 \\ 0.19 & 0.47 & 0.34 \\ 0.19 & 0.5 & 0.31 \end{pmatrix} \\
&\vdots \\
P^{100} &= \begin{pmatrix} 0.18446602 & 0.48543689 & 0.33009709 \\ 0.18446602 & 0.48543689 & 0.33009709 \\ 0.18446602 & 0.48543689 & 0.33009709 \end{pmatrix} \\
P^{101} &= \begin{pmatrix} 0.18446602 & 0.48543689 & 0.33009709 \\ 0.18446602 & 0.48543689 & 0.33009709 \\ 0.18446602 & 0.48543689 & 0.33009709 \end{pmatrix} \qquad \vdots
\end{aligned}
$$

We notice two things from this:

- if each row is identical, then is matters not which state we begin in;
- after a some number time steps, $P^n = P^{n+1} = \lim_{n\to\infty} P^n$.

When this situation occurs, the system is said to be in *steady-state*. The rows of resultant matrix $\lim_{n\to\infty} P^n$ is called the long-run probabilities, or steady-state probabilities of the Markov chain.

DTMC Steady-State

We can find the steady-state probabilities, π of a Markov chain by solving:

$$\underline{\pi} = \underline{\pi} P \tag{2.1}$$

along with the extra constraint enforcing $\underline{\pi}$ to be a valid probability vector:

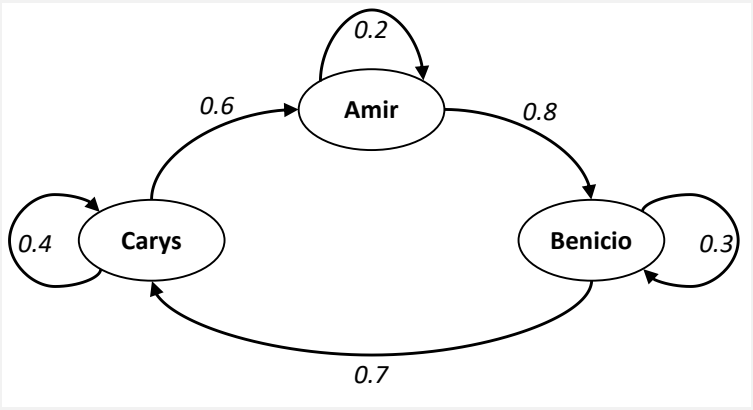
$$\sum \underline{\pi} = 1$$

[1] This will be discussed more in Section 2.4

We can solve this using various linear algebraic methods.

Example 4 *Three children, Amir, Benicio, and Carys, are playing pass-the-parcel. They are sat in alphabetical order clockwise. At each beat of the music, the child will pass the parcel to their left clockwise, or keep the parcel for one more beat. Amir will keep the parcel with probability 0.2, Benicio will keep the parcel with probability 0.3, and Carys will keep the parcel with probability 0.4.*
After enough time has passed to assume steady-state, if we randomly switch of the music, what is the probability that each child will be holding the parcel?

Solution to Example 4 *Drawing the Markov chain will help:*



We have:

$$P = \begin{pmatrix} 0.2 & 0.8 & 0 \\ 0 & 0.3 & 0.7 \\ 0.6 & 0 & 0.4 \end{pmatrix}$$

and we wish to find $\underline{\pi}$ *by solving:*

$$\underline{\pi} = \underline{\pi} P$$
$$\sum \underline{\pi} = 1$$

Yielding

$$\begin{aligned} \pi_A &= 0.2\pi_A + 0.6\pi_C \\ \pi_B &= 0.8\pi_A + 0.3\pi_B \\ \pi_C &= 0.7\pi_B + 0.4\pi_C \\ 1 &= \pi_A + \pi_B + \pi_C \end{aligned}$$

Solution to Example 4 (continuing from p. 20) *The first two equations gives us all components in terms of π_C:*

$$\begin{aligned}
\pi_A &= \frac{0.6}{1-0.2}\pi_C = \frac{3}{4}\pi_C \\
\pi_B &= \frac{0.8}{1-0.3}\pi_A = \frac{8}{7}\pi_A \\
&= \frac{8}{7}\left(\frac{3}{4}\pi_C\right) \\
&= \frac{6}{7}\pi_C
\end{aligned}$$

Substituting into the final equation gives:

$$\begin{aligned}
\pi_A + \pi_B + \pi_C &= 1 \\
\frac{3}{4}\pi_C + \frac{6}{7}\pi_C + \pi_C &= 1 \\
\pi_C\left(\frac{3}{4} + \frac{6}{7} + 1\right) &= 1 \\
\frac{73}{28}\pi_C &= 1 \\
\pi_C &= \frac{28}{73}
\end{aligned}$$

Now as we have all components in terms of π_C:

$$\begin{aligned}
\pi_C &= \frac{28}{73} \\
\pi_B &= \frac{6}{7}\pi_C = \frac{6}{7}\left(\frac{28}{73}\right) = \frac{24}{73} \\
\pi_A &= \frac{3}{4}\pi_C = \frac{3}{4}\left(\frac{28}{73}\right) = \frac{21}{73}
\end{aligned}$$

And so the probabilities that each child will be left holding the parcel is $\underline{\pi} = ({}^{21}/_{73}, {}^{24}/_{73}, {}^{28}/_{72})$.

2.3 Continuous-Time Markov Chains

Discrete-Time Markov chains are called that because the time domain takes on discrete time steps, $n \in \mathbb{N}$. We can also define Markov chains, and stochastic processes in general, when the time domain is continuous, $t \in \mathbb{R}^+$.

Continuous-Time Stochastic Process

A continuous-time stochastic process is a random vector, whose components are indexed by time. It can also be thought of as a family of random variables:

$$\{X_t,\ t \in \mathbb{R}^+\}$$

A continuous-time Markov chain is a continuous-time stochastic process, where the outcome of each X_t is an element of a countable set of states, called the *state space*, and that satisfies the *Markov property*. In continuous-time case, that becomes:

Markov Property (Continuous-Time)

The Markov property states that $X_{t+\tau}$ depends only on X_t, and not on any previous states, X_s, $s < t$. That is:

$$\mathbb{P}(X_{t+\tau} = x_{t+\tau} \mid X_s = x_s\ \forall\ s \leq t) = \mathbb{P}(X_{t+\tau} = x_{t+\tau} \mid X_t = x_t)$$

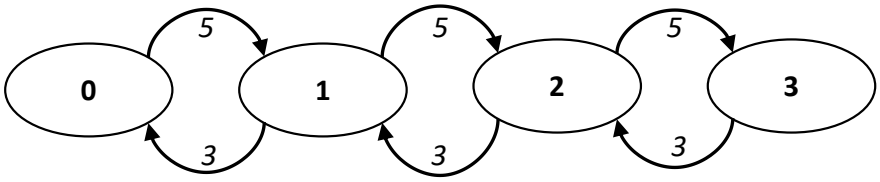
For a continuous-time Markov chain, the state space is defined identically to the discrete-time Markov chain. We assume that the system changes state randomly in time, so we let $\underline{\pi}(t)$ denote the probability vector of the system at time t. Therefore, instead of a transition probability matrix, we now have a *transition rate matrix*, or an *infinitesimal generator matrix* Q, defined such that:

$$\frac{d}{dt}\underline{\pi}(t) = \underline{\pi}(t)Q \tag{2.2}$$

where:

- the entry Q_{ij} denotes the rate at which the system moves from state s_i to state s_j, if $i \neq j$, and
- the entry $-Q_{ii}$ is represents the rate at which the system moves out of state s_i.

As an example, consider a charity book shop, which regularly receives donation of The Da Vinci Code by Dan Brown, and also regularly sells the book. They receive so many that they will only keep a maximum of three in stock at any point, and will refuse any more donations once reached. Let n be the number of copies of The Da Vinci Code in the shop. On average they receive 5 copies in donations per week, and sell 3 copies per week. This Markov chain is illustrated below:



The state space is $S = \{0, 1, 2, 3\}$, and the transition rate matrix is given by:

$$Q = \begin{pmatrix} -5 & 5 & 0 & 0 \\ 3 & -8 & 5 & 0 \\ 0 & 3 & -8 & 5 \\ 0 & 0 & 3 & -3 \end{pmatrix}$$

Then from Equation 2.2:

$$\begin{aligned} \frac{d}{dt}\pi_0(t) &= -5\pi_0(t) + 3\pi_1(t) \\ \frac{d}{dt}\pi_1(t) &= 5\pi_0(t) - 8\pi_1(t) + 3\pi_2(t) \\ \frac{d}{dt}\pi_2(t) &= 5\pi_1(t) - 8\pi_2(t) + 3\pi_3(t) \\ \frac{d}{dt}\pi_3(t) &= 5\pi_2(t) - 3\pi_3(t) \end{aligned}$$

These differential equations give the transient analysis of the Markov chain, however this is not considered further in this course.

Holding Times

Holding times are the amount of time a continuous-time Markov chain remains in a particular state before changing state. The times to change state are random in time, relying only on the current state and no previous history. Therefore, the times between state changes can be described by an Exponential distribution.

In particular, let T_{ij} be the time spent in s_i before changing to state s_j, then $T_{ij} \sim \text{Expon}(Q_{ij})$. Now consider the time it spends there before moving on to any other state, that is the holding time, $\widetilde{T}_i$:

CTMC Holding Times

Theorem: The holding time in state i is Exponentially distributed with rate $\sum_{i \neq j} Q_{ij}$. That is $\widetilde{T}_i \sim \text{Expon}\left(\sum_{i \neq j} Q_{ij}\right)$.

Proof: We know that $\widetilde{T}_i = \min_{i \neq j} T_{ij}$. Consider the CDF of $\widetilde{T}_i$:

$$\begin{aligned}
\mathbb{P}(\widetilde{T}_i > t) &= \mathbb{P}\left(\min_{i \neq j} T_{ij} > t\right) \\
&= \mathbb{P}\left(\bigcap_{i \neq j} T_{ij} > t\right) \\
&= \prod_{i \neq j} \mathbb{P}\left(T_{ij} > t\right) \\
&= \prod_{i \neq j} e^{-Q_{ij} t} \\
&= e^{-\sum_{i \neq j} Q_{ij} t}
\end{aligned}$$

therefore

$$\widetilde{T}_i \sim \text{Expon}\left(\sum_{i \neq j} Q_{ij}\right).$$

Now we can verify and interpret the Markov property: the probability of being in a particular state at time $t + \tau$ relies only on the state at time t, and not on any of the states visited before time t, nor on how long the system has already been in its current state. This comes from the memorylessness of the Exponential distribution.

Steady-State Analysis

In general, the transition rate matrix of a continuous time Markov chain gives the rate at which the states change, that is $\frac{d}{dt}\pi(t) = \pi(t)Q$. We have said that the transient analysis this is not considered in this course. However we can say something about the system in steady-state. In steady-state, we know that $\frac{d}{dt}\pi(t) = 0$, that is the state vector $\pi(t)$ does not change any more. Therefore:

CTMC Steady-State

We can find the steady-state probabilities, π of a Markov chain by solving:

$$\underline{\pi}Q = \underline{0} \tag{2.3}$$

along with the extra constraint enforcing $\underline{\pi}$ to be a valid probability vector:

$$\sum \underline{\pi} = 1$$

Example 5 *Consider the Markov chain defined above concerning the number of copies of the Da Vinci Code a charity shop stocks. Here* $S = \{0, 1, 2, 3\}$ *and*

$$Q = \begin{pmatrix} -5 & 5 & 0 & 0 \\ 3 & -8 & 5 & 0 \\ 0 & 3 & -8 & 5 \\ 0 & 0 & 3 & -3 \end{pmatrix}$$

What are the long run steady-state probabilities of being in each state?

Solution to Example 5 *Solving* $\underline{\pi}Q = \underline{0}$ *along with* $\sum \underline{\pi} = 1$ *gives the following system of equations:*

$$-5\pi_0 + 3\pi_1 = 0 \tag{2.4}$$
$$5\pi_0 - 8\pi_1 + 3\pi_2 = 0 \tag{2.5}$$
$$5\pi_1 - 8\pi_2 + 3\pi_3 = 0 \tag{2.6}$$
$$5\pi_2 - 3\pi_3 = 0 \tag{2.7}$$
$$\pi_0 + \pi_1 + \pi_2 + \pi_3 = 1 \tag{2.8}$$

This is a system of five linear equations of four unknowns. We can use any method we like of solving these. Here let's use three of the first four equations to put every variable in terms of π_0*, then use the final equation to find the value of* π_0*:*
For π_1 *in terms of* π_0 *consider Equation 2.4:*

$$\begin{aligned} -5\pi_0 + 3\pi_1 &= 0 \\ 3\pi_1 &= 5\pi_0 \\ \pi_1 &= \frac{5}{3}\pi_0 \end{aligned}$$

Solution to Example 5 (continuing from p. 25) *For π_2 in terms of π_0 consider Equation 2.5 and substitute in $\pi_1 = \frac{5}{3}\pi_0$:*

$$\begin{aligned} 5\pi_0 - 8\pi_1 + 3\pi_2 &= 0 \\ 3\pi_2 &= -5\pi_0 + 8\pi_1 \\ 3\pi_2 &= -5\pi_0 + 8\left(\frac{5}{3}\pi_0\right) \\ 3\pi_2 &= -\frac{15}{3}\pi_0 + \frac{40}{3}\pi_0 \\ 3\pi_2 &= \frac{25}{3}\pi_0 \\ \pi_2 &= \frac{25}{6}\pi_0 \end{aligned}$$

For π_3 in terms of π_0 consider Equation 2.7 and substitute in $\pi_2 = \frac{25}{6}\pi_0$:

$$\begin{aligned} 5\pi_2 - 3\pi_3 &= 0 \\ -3\pi_3 &= -5\pi_2 \\ -3\pi_3 &= -5\left(\frac{25}{6}\pi_0\right) \\ -3\pi_3 &= -\frac{125}{6}\pi_0 \\ \pi_3 &= \frac{125}{18}\pi_0 \end{aligned}$$

Finally, use Equation 2.8 to find the value of π_0:

$$\begin{aligned} \pi_0 + \pi_1 + \pi_2 + \pi_3 &= 1 \\ \pi_0 + \frac{5}{3}\pi_0 + \frac{25}{6}\pi_0 + \frac{125}{18}\pi_0 &= 1 \\ \left(1 + \frac{5}{3} + \frac{25}{6} + \frac{125}{18}\right)\pi_0 &= 1 \\ \left(\frac{18}{18} + \frac{30}{18} + \frac{75}{18} + \frac{125}{18}\right)\pi_0 &= 1 \\ \frac{248}{18}\pi_0 &= 1 \\ \frac{124}{9}\pi_0 &= 1 \\ \pi_0 &= \frac{9}{124} \end{aligned}$$

Solution to Example 5 (continuing from p. 26) *And substituting this value into all the other relations gives:*

$$\begin{aligned}\pi_0 &= \frac{9}{124} \\ \pi_1 &= \frac{5}{3} \times \frac{9}{124} = \frac{15}{124} \\ \pi_2 &= \frac{25}{6} \times \frac{9}{124} = \frac{75}{248} \\ \pi_3 &= \frac{125}{18} \times \frac{9}{124} = \frac{125}{248}\end{aligned}$$

Therefore $\underline{\pi} = (^{9}/_{124}, ^{15}/_{124}, ^{75}/_{248}, ^{125}/_{248})$.

Embedded Markov Chain

An embedded Markov chain of a continuous-time Markov chain is a discrete-time Markov chain that describes the same system. We have:

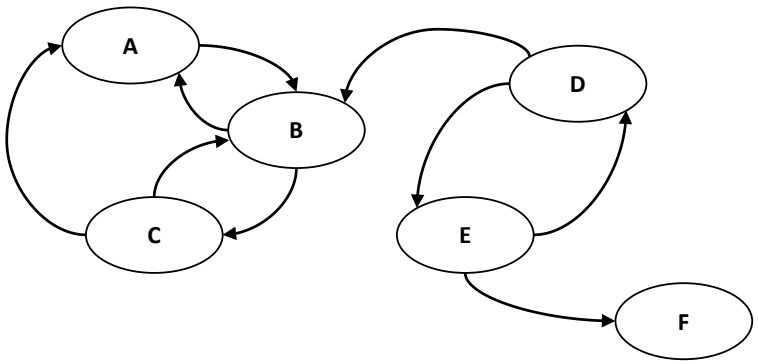
$$P = Q\Delta t + \mathbb{I}$$

where $\mathbb{I}$ is an identity matrix of the same size as Q, and Δt is the length of time one time step is defined as in the embedded chain. We need to take Δt to be sufficiently small, and so generally we can take:

$$\Delta t = \frac{1}{\max_i |Q_{ii}|}$$

2.4 Classifying States

States can be classified according to their reachability. Consider the Markov chain below:



Observations:

- Once the system is in states A, B, or C, then is can only ever reach state A, B, or C in the future;
- Once the system is in state F, then it will remain in state F for evermore;
- If the system is in states D or E, it will eventually make its way into states A, B, C, or F, and won't return to state D or E.

There is a language we can use to describe these scenarios.

Accessible and Communicating States

Definition: State s_j is said the be *accessible* from state s_i if starting in state s_i it is possible to enter state s_j at some time. We write $s_i \rightsquigarrow s_j$.

Definition: Two states s_i and s_j that are accessible from each other are said to *communicate*. We write $s_i \leftrightsquigarrow s_j$.

Note that the relation $\leftrightsquigarrow$ is an equivalence relation, so:

- $s_i \leftrightsquigarrow s_i$ for all s_i,
- if $s_i \leftrightsquigarrow s_j$ then $s_j \leftrightsquigarrow s_i$,
- if $s_i \leftrightsquigarrow s_j$ and $s_j \leftrightsquigarrow s_k$, then $s_i \leftrightsquigarrow s_k$.

We can split the state space into equivalence classes, such that two states s_i and s_j belong to the same class if and only if $s_i \leftrightsquigarrow s_j$. These are called **irreducible classes**.

In the example above, there are three classes: $\mathcal{C}_1 = \{A, B, C\}$, $\mathcal{C}_2 = \{D, E\}$ and $\mathcal{C}_3 = \{F\}$. Each of these have different properties:

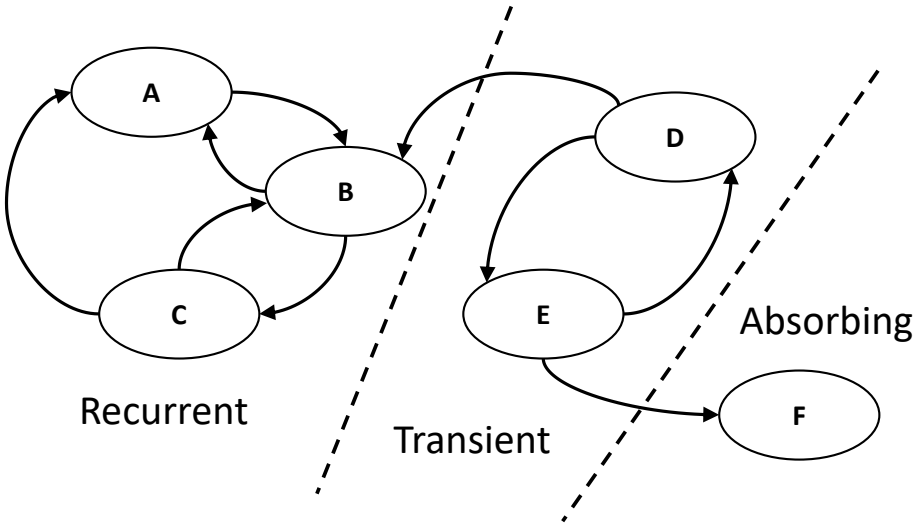
- Only states in $\mathcal{C}_1$ are accessible from states in $\mathcal{C}_1$. Similarly with $\mathcal{C}_3$. We call these classes *closed*.
- States outside $\mathcal{C}_2$ are accessible from state in $\mathcal{C}_2$. We call this class *not closed*.

Transient, Recurrent, and Absorbing States

Definition:

- States belonging to a not closed irreducible class are called *transient states*.
- States belonging to a closed irreducible class are called *recurrent states*.
- Recurrent states in an irreducible class on its own are called *absorbing states*.

Therefore we can classify the states in our example:



Periodicity

For discrete-time Markov chains there is an extra consideration when classifying states: periodicity. Briefly, that means that a state is periodic if the chain can return to the state only at time steps of multiples of some integer larger than 1.

Consider the frog-leaping example: beginning on an odd-numbered lily-pad, you can only reach even-numbered lily-pads in the next time step, and vice versa. Therefore, each state has period-2.

A state with period-1 is called *aperiodic*.

Existence of Steady-States

Discrete-Time Markov Chains: When a discrete-time Markov chain consists of one irreducible class, and all states are aperiodic, then it is guaranteed to have a unique steady-state distribution.

Continuous-Time Markov Chains: A continuous-time Markov chain is guaranteed to have a unique steady-state distribution if its embedded Markov chain has a unique steady-state distribution.

Chapter 3

Queueing Theory

Learning Outcomes:

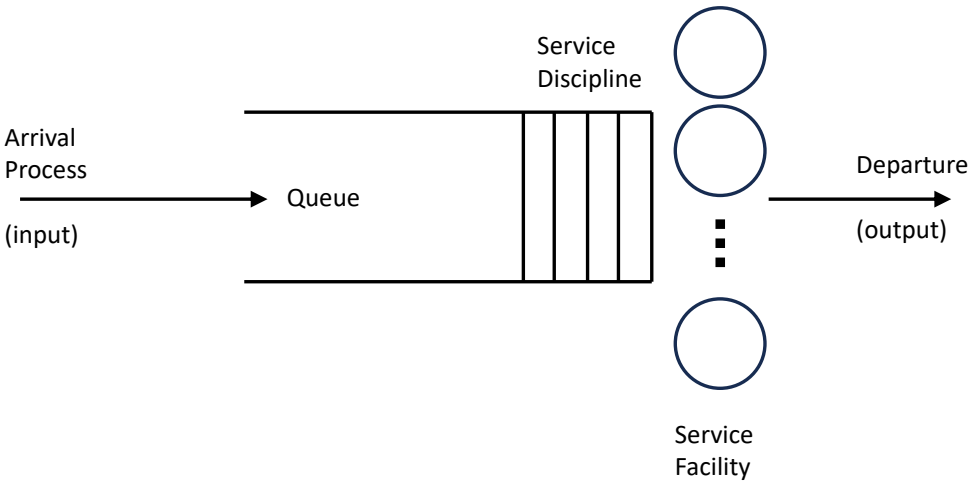
- Be able to understand the language and notation of queueing theory;
- Be able to derive steady-state distributions for capacitated queues;
- Be able to derive steady-state distributions for infinite queues;
- Be able to perform some cost analysis and scenario comparisons for queueing systems.

3.1 Introduction

When a sequence of customers arrive at a service facility requiring some type of service to be performed on them, we say that this is a queueing system. Usually, the customers arriving to the service facility have to wait for a server to be free before receiving their own service, hence waiting in a queue.

Examples of queueing systems include customers arriving to a shop counter; aeroplanes waiting to land on a runway; or calls waiting to be processed at a switchboard. In whichever situation we are modelling, we use the the term **customer** to refer to the entity requiring service; **service** to refer to the amount of time a customer uses a resource; and **server** to refer to the resource itself.

The diagram below shows a queueing system:

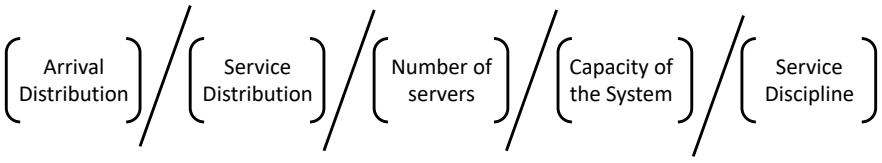


Five components are labelled:

- **Arrival process:** This describes the manner in which customers enter the system, or join the queue. It is standard to describe this by the distribution of the *inter-arrival times*, that is the times between any two consecutive arrivals.
- **Queue:** This refers to the area in which customers wait for service, sometimes called a buffer. There can be an infinite or a finite capacity on the queueing area. In a finite buffer, once full, newly arriving customers are turned away.
- **Service Facility:** This refers to the resources available to provide the service, and the length of time spend receiving the service.
- **Service Discipline:** This refers to the order in which waiting customers are chosen to receive service.
- **Departure:** This refers to how customers exit the system after receiving service. Generally this would be immediate departure after service.

3.2 Kendall's Notation

Developed by D.G. Kendall in 1953, and expanded on by A.M. Lee in 1966, this notation allows us to describe the main properties of a queue in shorthand. It is in standard use in operational research. There are five components to consider:



For example: $M/G/3/\infty/LIFO$. Each element between the slashes describes the component it refers to. The table below summarises some values they can take:

Component	Element	Explanation
Arrival distribution	M	*Markovian*. Exponentially distributed inter-arrival times.
	M^x	*Markovian with batch arrivals*. When an arrival occurs, x customers arrive at the same time.
	D	*Deterministic* or degenerate. Inter-arrival times take on a fixed value.
	E_k	*Erlang*. Inter-arrival times have an Erlang distribution with parameter k. That is the sum of k independent Exponentially distributed variables.
	G	*General*. Inter-arrival times take on any general probability distribution.
Service distribution	M	*Markovian*. Exponentially distributed service times.
	D	*Deterministic* or degenerate. Service times take on a fixed value.
	E_k	*Erlang*. Service times have an Erlang distribution with parameter k. That is the sum of k independent Exponentially distributed variables.
	G	*General*. Service times take on any general probability distribution.
Number of servers	$c = 1, 2, 3 \ldots$	An integer number of servers.
	∞	An infinite number of servers.
Capacity of the system	$K = 1, 2, 3 \ldots;\ K \geq c$	An integer capacity greater than the number of servers.
	∞	An infinite capacity.
Service discipline	$FIFO$	*First-in-first-out*.
	$LIFO$	*Last-in-first-out*.
	$SIRO$	*Service-in-random-order*.
	PQ	*Priority queues*. Some customers have priority over others and are chosen for service before others.
	PS	*Processor sharing*. All customers enter service immediately when entering the system, and their service rate slows the more customers are present.

Some examples:

- $M/D/3/\infty$/FIFO:
  Markovian arrivals, deterministic service times, 3 servers, infinite capacity, first-in-first-out service discipline.
- $G/G/\infty/\infty$/PS:
  General arrivals, general service times, infinite servers, infinite capacity, processor sharing service discipline.
- $M/M/1/12$/SIRO:
  Markovian arrivals, Markovian service times, single server, maximum capacity of 12, service-in-random-order service discipline.

Generally, if only the first three are given, then we assume infinite capacity and FIFO service discipline:

- $D/E_4/2$:
  Deterministic arrivals, service times distributed with an Erlang distribution with parameter 4, 2 servers, infinite capacity, first-in-first-out service discipline.
- $M/M/1$:
  Markovian arrivals, Markovian service times, single server, infinite capacity, first-in-first-out service discipline.

3.3 Measures of Interest

When analysing a queueing system, no matter what the arrival process, service time distribution, or service discipline, there are some measures of interest that are useful for us to analyse. Here we will talk about infinite capacity queues.

First, consider that customers arrive to the queue at a rate of λ per time unit, and that customers are served at a rate μ per time unit. That is, whatever the inter-arrival time distribution it has mean $1/\lambda$, and whatever the service time distribution it has mean $1/\mu$. Consider also that there are c parallel servers.

Then:

Traffic Intensity

The traffic intensity of the queue is:

$$\rho = \frac{\lambda}{c\mu} \tag{3.1}$$

The traffic intensity ρ is an extremely useful measure of a queue. The numerator λ is the rate at which customers join the queue, and the denominator $c\mu$ is the maximum effective rate at which customers leave the service facility (when the queue isn't busy, the effective service rate will be less than this). If the numerator is larger than the denominator, then there are more customers entering the queue than can ever be served, and the queue size will grow and grow. Therefore:

Steady-State Condition

- If $\rho \geq 1$ then the queue is expected to grow indefinitely;
- If $\rho < 1$ then the system is *stable*, the queue size doesn't grow indefinitely.

From now on we will only consider stable systems, where we expect the queue to reach some form of steady-state, like a Markov chain. Other variables of interest are:

- L: the average number of customers in the system in steady-state;
- L_q: the average number of customers in the queue in steady-state;
- W: the average time customers spend in the system in steady-state;
- W_q: the average time customers spend in the queue in steady-state;
- P_n: the probability of there being n customers in the system in steady-state.

We have a notorious result by J. Little in 1954 (and later proved in 1961) that relates two of these for any stable queue, of any service discipline and any inter-arrival time and service time distributions. It is an astonishing result that such a simple formula can apply to such an infinitely wide variety of queues:

Little's Laws

$$L = \lambda W \tag{3.2}$$
$$L_q = \lambda W_q \tag{3.3}$$

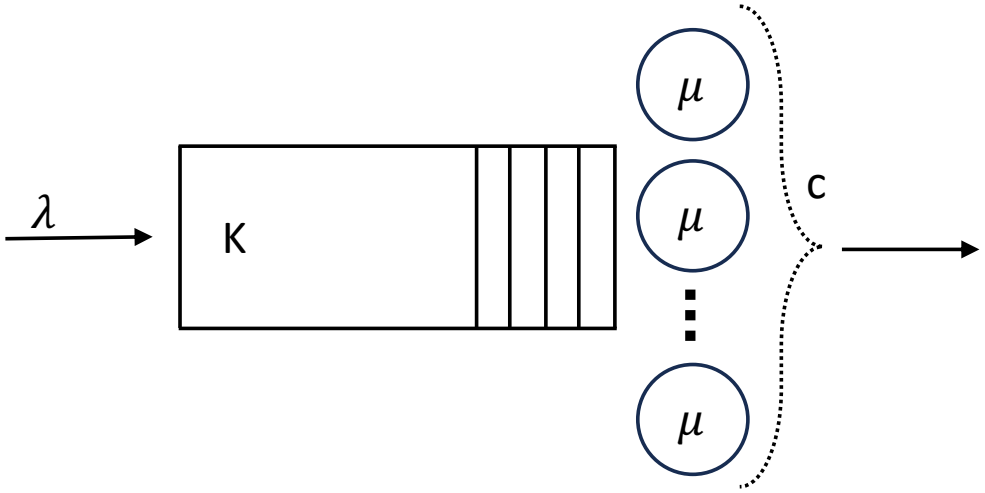
Some other results can be easily found heuristically, for example:

- $W_q = W - 1/\mu$, the average time spent in the queue is equal to the average time spent in the system minus the average service time.
- the traffic intensity ρ is also the average utilisation of a server in steady-state, that is the proportion of the time a server is busy.
- $L_q = \lambda W_q = \lambda\left(W - 1/\mu\right) = \lambda W - \rho$.
- $P_0 = 1 - \rho$, the probability that no-one is in the system is the probability that the server isn't busy, that is 1 minus the probability of the server being busy.

We will now derive expressions for many of these measures for a Markovian queues. We will begin with a finite capacity system as this is easier to deal with, and then consider an infinite capacity system.

3.4 An $M/M/c/K$/FIFO Queue

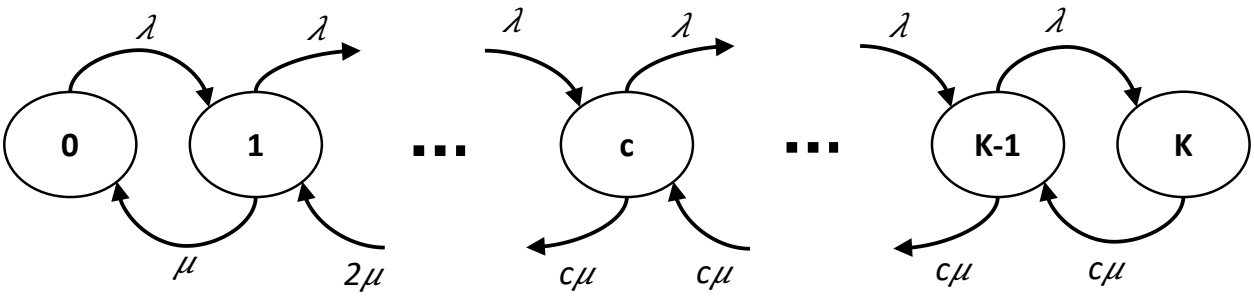
First consider an $M/M/c/K$/FIFO queue with arrival rate λ and service rate μ. This is illustrated below:



Note that in this diagram, there is a vertical line closing the opening of the queue, this indicates a capacitated system. This can be analysed as a continuous-time Markov chain. The Markovian arrivals and services correspond to the Exponentially distributed holding times of the Markov states, where the memoryless property holds. Now:

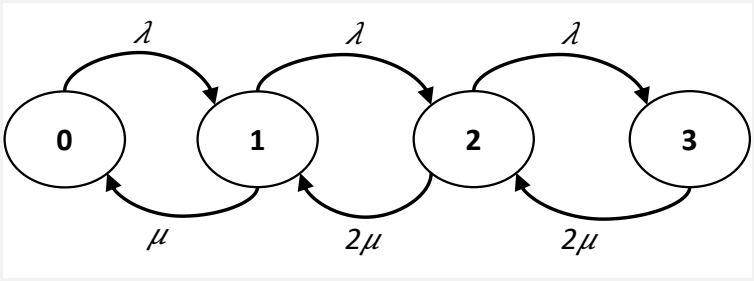
- Let $S = \{0, 1, 2, \ldots, K-1, K\}$ be the state space, representing the number of customers in the system.
  - 0 represents no customers in the system;
  - $i \in \{1, \ldots, c\}$ represent i in service, and none queueing;
  - $i \in \{c+1, \ldots K\}$ represent c in service, and $i - c$ customers waiting.
- Let Q_{ij} be the rate at which the system moves from state i to state j. Then:
  - to go from state i to $i+1$ means a customer has arrived. This happens with rate λ.
  - to go from state i to $i-1$ means a customer is being served. When there are $i \leq c$ customer in service, this happens with rate $i\mu$, due to the superposition of Poisson processes. When there are $i > c$ customers in the system, this happens with rate $c\mu$.

This Markov chain is illustrated below:



Example 6 *Consider a $M/M/2/3$/FIFO queue.*
(a) Draw the continuous-time Markov chain and write down the transition matrix.
(b) Find the steady-state probabilities.
(c) If $\lambda = 3$ and $\mu = 5$, find ρ, L, W, L_q, and W_q.

Solution to Example 6 *In turn:*
(a) We get:



with $S = \{0, 1, 2, 3\}$ and:

$$Q = \begin{pmatrix} -\lambda & \lambda & 0 & 0 \\ \mu & -(\lambda+\mu) & \lambda & 0 \\ 0 & 2\mu & -(\lambda+2\mu) & \lambda \\ 0 & 0 & 2\mu & -2\mu \end{pmatrix}$$

(b) Solving $\underline{P}Q = 0$ along with $\sum \underline{P} = 1$ we get:

$$\begin{aligned} \lambda P_0 &= \mu P_1 \\ (\lambda+\mu)P_1 &= \lambda P_0 + 2\mu P_2 \\ (\lambda+2\mu)P_2 &= \lambda P_1 + 2\mu P_3 \\ 2\mu P_3 &= \lambda P_2 \\ P_0 + P_1 + P_2 + P_3 &= 1 \end{aligned}$$

Solution to Example 6 (continuing from p. 36) *Now to solve we'll put every variable in terms of P_0, then use the final equation to find the value of P_0. First P_1:*

$$\begin{aligned}\lambda\mu P_1 &= P_0\\ P_1 &= \frac{\lambda}{\mu}P_0\end{aligned}$$

Now P_2:

$$\begin{aligned}\lambda P_0 + 2\mu P_2 &= (\lambda+\mu)P_1\\ \lambda P_0 + 2\mu P_2 &= (\lambda+\mu)\frac{\lambda}{\mu}P_0\\ 2\mu P_2 &= \left(\frac{\lambda^2}{\mu} + \lambda - \lambda\right)P_0\\ P_2 &= \frac{\lambda^2}{2\mu^2}P_0\end{aligned}$$

Now P_3:

$$\begin{aligned}2\mu P_3 &= \lambda P_2\\ P_3 &= \frac{\lambda}{2\mu}P_2\\ P_3 &= \frac{\lambda}{2\mu}\left(\frac{\lambda^2}{2\mu^2}\right)P_0\\ P_3 &= \frac{\lambda^3}{4\mu^3}P_0\end{aligned}$$

And now:

$$\begin{aligned}P_0 + P_1 + P_2 + P_3 &= 1\\ P_0\left(1 + \frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \frac{\lambda^3}{4\mu^3}\right) &= 1\\ P_0 &= \frac{1}{\left(1 + \frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \frac{\lambda^3}{4\mu^3}\right)}\end{aligned}$$

Solution to Example 6 (continuing from p. 37) *Therefore:*

$$P_0 = \frac{1}{\left(1 + \frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \frac{\lambda^3}{4\mu^3}\right)}$$
$$P_1 = \frac{\lambda}{\mu\left(1 + \frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \frac{\lambda^3}{4\mu^3}\right)}$$
$$P_2 = \frac{\lambda^2}{2\mu^2\left(1 + \frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \frac{\lambda^3}{4\mu^3}\right)}$$
$$P_3 = \frac{\lambda^3}{4\mu^3\left(1 + \frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \frac{\lambda^3}{4\mu^3}\right)}$$

(c) *Substituting in* $\lambda = 3$ *and* $\mu = 5$ *gives:* $P_0 = \frac{500}{917}$, $P_1 = \frac{300}{917}$, $P_2 = \frac{90}{917}$, *and* $P_3 = \frac{27}{917}$. *Then:*

$$L = \sum_{i=0}^{3} iP_i = \frac{300}{917} + \left(2 \times \frac{90}{917}\right) + \left(3 \times \frac{27}{917}\right) = \frac{561}{917}$$

$$W = \frac{1}{\lambda}L = \frac{1}{3} \times \frac{561}{917} = \frac{187}{917}$$

$$W_q = W - \frac{1}{\mu} = \frac{187}{917} - \frac{1}{5} = \frac{18}{4585}$$

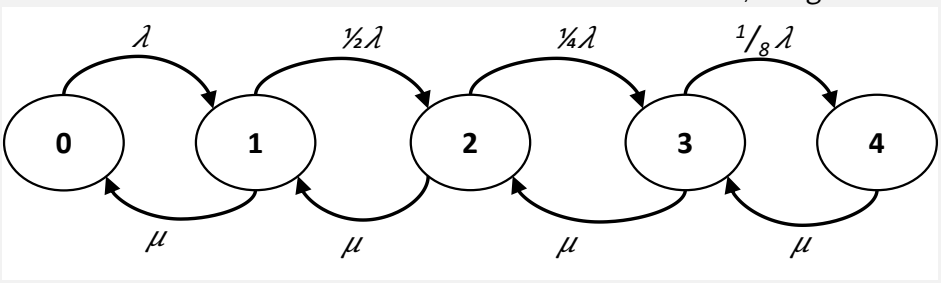
$$L_q = \lambda W_q = 3 \times \frac{18}{4585} = \frac{54}{4585}$$

Another example:

Example 7 *Consider an* $M/M/1/4$ *queue, where customers are impatient. If an arriving customer arrives when there are* n *people already in the system, then they will only join the queue with probability* $\frac{1}{2^n}$. *Find the steady-state probabilities when* $\lambda = 2$ *and* $\mu = 1$. *What proportion of arriving customers do not join the queue?*

Note that this kind of behaviour is called *baulking*.

Solution to Example 7 *Due to Poisson thinning, this is still a continuous-time Markov chain, where the arrival rates are 'thinned' as the states increase. Therefore, we get:*



solving $\underline{P}Q = 0$ *we get:*

$$\begin{aligned}
\lambda P_0 &= \mu P_1 \\
(1/2\lambda + \mu)\, P_1 &= \lambda P_0 + \mu P_2 \\
(1/4\lambda + \mu)\, P_2 &= 1/2\lambda P_1 + \mu P_3 \\
(1/8\lambda + \mu)\, P_3 &= 1/4\lambda P_2 + \mu P_4 \\
\mu P_4 &= 1/8\lambda P_3
\end{aligned}$$

Rearranging in a similar way to before gives:

$$P_1 = \frac{\lambda}{\mu}P_0 \qquad P_2 = \frac{\lambda^2}{2\mu^2}P_0 \qquad P_3 = \frac{\lambda^3}{8\mu^3}P_0 \qquad P_4 = \frac{\lambda^4}{64\mu^4}P_0$$

giving:

$$\begin{aligned}
P_0 + P_1 + P_2 + P_3 + P_4 &= 1 \\
P_0\left(\frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \frac{\lambda^3}{8\mu^3} + \frac{\lambda^4}{64\mu^4}\right) &= 1 \\
P_0 &= \frac{1}{\frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \frac{\lambda^3}{8\mu^3} + \frac{\lambda^4}{64\mu^4}}
\end{aligned}$$

Substituting in $\lambda = 2$ *and* $\mu = 1$ *gives:*

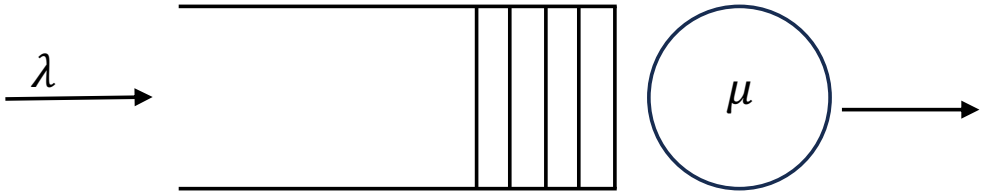
$$(P_0, P_1, P_2, P_3, P_4) = \left(\frac{4}{29}, \frac{8}{29}, \frac{8}{29}, \frac{8}{29}, \frac{1}{29}\right)$$

In each state the probability of baulking is: 0 *in state 0;* $1/2$ *in state 1;* $3/4$ *in state 2;* $7/8$ *in state 3; and* 1 *in state 4 (that is they are turned away as it is too full). Therefore:*

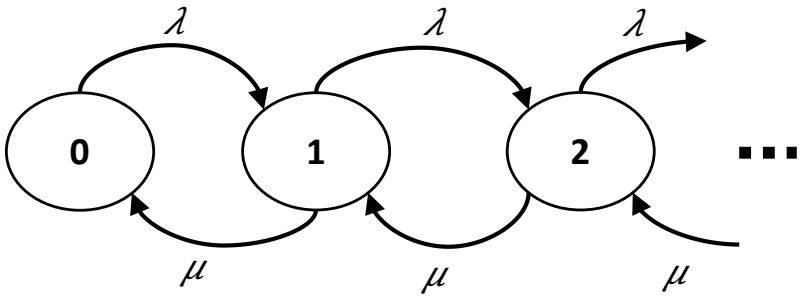
$$\begin{aligned}
\mathbb{P}(\textit{arriving customer baulking}) &= (1/2 \times 8/29) + (3/4 \times 8/29) + (7/8 \times 8/29) + (1 \times 1/29) \\
&= 18/29
\end{aligned}$$

3.5 An $M/M/1$ Queue

An $M/M/1$ queue, or an $M/M/1/\infty$/FIFO is different as there is now an infinite buffer.



This means that the corresponding continuous-time Markov chain has infinite state space.



That is:

$$Q = \begin{pmatrix} -\lambda & \lambda & 0 & 0 & \dots \\ \mu & -(\lambda+\mu) & \mu & 0 & \dots \\ 0 & \mu & -(\lambda+\mu) & \mu & \dots \\ 0 & 0 & \mu & -(\lambda+\mu) & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

The Markov chain is sometimes called a *birth-death process*. Now we know that steady-state will only exist if the system is stable, that is if $\rho = {}^{\lambda}/_{\mu} < 1$. A steady-state here will mean that there exists some integer N, such that for all $n > N$, $P_n < \epsilon$ for some arbitrarily small ϵ. Heuristically, if $\rho < 1$, then $\lambda < \mu$, then the rate at which the system moves up the chain is smaller than the rate at which the system moves down the chain, so we expect the P_i to decrease as i increases, and so this value N would exist.

We will have to analyse this by developing general expressions for P_i for each $i \in \mathbb{N}$.

As an aside, we will make use of two standard results, when $r < 1$:

$$S = \sum_{k=0}^{\infty} r^k = \frac{1}{1-r} \qquad Z = \sum_{k=0}^{\infty} k r^k = \frac{r}{(1-r)^2}$$

Assuming their convergence, the values of these limits can be found heuristically:

$$\begin{aligned} S &= 1 + r + r^2 + r^3 + \dots \\ rS &= r + r^2 + r^3 + r^4 + \dots \\ S - rS &= 1 \\ S(1-r) &= 1 \\ S &= \frac{1}{1-r} \end{aligned}$$

$$\begin{aligned} Z &= r + 2r^2 + 3r^3 + 4r^4 + \dots \\ rZ &= r^2 + 2r^3 + 3r^4 + 5r^4 + \dots \\ Z - rZ &= r + r^2 + r^3 + r^4 + \dots \\ Z - rZ &= rS \\ Z(1-r) &= \frac{r}{1-r} \\ Z &= \frac{r}{(1-r)^2} \end{aligned}$$

Now, solving $\underline{P}Q = 0$ gives:

$$\begin{aligned} \lambda P_0 &= \mu P_1 \\ (\lambda + \mu) P_1 &= \lambda P_0 + \mu P_2 \\ (\lambda + \mu) P_2 &= \lambda P_1 + \mu P_3 \\ &\vdots \\ (\lambda + \mu) P_k &= \lambda P_{k-1} + \mu P_{k+1} \\ &\vdots \end{aligned}$$

Taking our cue from Example 6 we will put each P_i in terms of P_0. For the first few terms we get:

$$\begin{aligned} P_1 &= \frac{\lambda}{\mu} P_0 = \rho P_0 \\ P_2 &= \frac{\lambda_2}{\mu^2} P_0 = \rho^2 P_0 \\ P_3 &= \frac{\lambda_3}{\mu^3} P_0 = \rho^3 P_0 \\ &\vdots \end{aligned}$$

So we might conclude that $P_k = \rho^k P_0$. If so, then as $\rho < 1$, then the P_i decrease as i increases, and $\lim_{n\to\infty} P_n = 0$, implying stability.

$M/M/1$ Steady-State Probabilities

Theorem: An $M/M/1$ queue has steady-state probabilities $P_k = \rho^k P_0$ for all $k \in \mathbb{N}$.
Proof: By induction. Base case P_1:

$$\begin{aligned} \mu P_1 &= \lambda P_0 \\ P_1 &= \frac{\lambda}{\mu} P_0 \\ P_1 &= \rho P_0 \end{aligned}$$

Assume that $P_j = \rho^j P_0$ for all $j \leq k$, what about P_{k+1}?

$$\begin{aligned} \lambda P_{k-1} + \mu P_{k+1} &= (\lambda + \mu) P_k \\ \lambda \rho^{k-1} P_0 + \mu P_{k+1} &= (\lambda + \mu) \rho^k P_0 \\ \mu P_{k+1} &= (\lambda + \mu) \rho^k P_0 - \lambda \rho^{k-1} P_0 \\ \mu P_{k+1} &= \lambda \rho^k P_0 + \mu \rho^k P_0 - \lambda \rho^{k-1} P_0 \\ P_{k+1} &= \frac{\lambda}{\mu} \rho^k P_0 + \rho^k P_0 - \frac{\lambda}{\mu} \rho^{k-1} P_0 \\ P_{k+1} &= \rho^{k+1} P_0 + \rho^k P_0 - \rho^k P_0 \\ P_{k+1} &= \rho^{k+1} P_0 \end{aligned}$$

which completes the proof.

How about the empty probability P_0?

$M/M/1$ Empty Probability

Theorem: An $M/M/1$ queue has empty probability $P_0 = 1 - \rho$.
Proof: We know that $P_k = \rho^k P_0$ for all $k \in \mathbb{N}$. We also know that the state probabilities sum to one. Therefore:

$$\begin{aligned} \sum_{k=0}^{\infty} P_k &= 1 \\ P_0 + P_1 + P_2 + P_3 + \ldots &= 1 \\ P_0 \left(1 + \rho + \rho^2 + \rho^3 + \ldots\right) &= 1 \\ P_0 \left(\frac{1}{1-\rho}\right) &= 1 \\ P_0 &= 1 - \rho \end{aligned}$$

from the convergence of the geometric series, as $\rho < 1$, which completes the proof.

And how about the other measures of interest, L, W, W_q and L_q?

$M/M/1$ Performance Measures

Theorem: For an $M/M/1$ queue we have:

$$L = \frac{\rho}{1-\rho} \qquad W = \frac{1}{\mu - \lambda} \qquad W_q = \frac{\rho}{\mu(1-\rho)} \qquad L_q = \frac{\rho^2}{1-\rho}$$

Proof: First consider L:

$$\begin{aligned} L &= \sum_{k=0}^{\infty} kP_k = (1-\rho)\sum_{k=0}^{\infty} k\rho^k \\ &= (1-\rho)\frac{\rho}{(1-\rho)^2} = \frac{\rho}{1-\rho} \end{aligned}$$

Now W from Little's law:

$$\begin{aligned} W &= \frac{1}{\lambda}L = \frac{1}{\lambda}\frac{\rho}{1-\rho} \\ &= \frac{1/\mu}{1-\rho} = \frac{1}{\mu - \lambda} \end{aligned}$$

And we get W_q by subtracting the expected service time:

$$\begin{aligned} W_q &= W - \frac{1}{\mu} \\ &= \frac{1}{\mu - \lambda} - \frac{1}{\mu} \\ &= \frac{\mu}{\mu(\mu-\lambda)} - \frac{\mu-\lambda}{\mu(\mu-\lambda)} \\ &= \frac{\lambda}{\mu(\mu-\lambda)} = \frac{\rho}{\mu(1-\rho)} \end{aligned}$$

And finally L_q again by Little's law:

$$\begin{aligned} L_q &= \lambda W_q \\ &= \lambda\frac{\rho}{\mu(1-\rho)} \\ &= \frac{\rho^2}{(1-\rho)} \end{aligned}$$

which completes the proof.

Example 8 *Consider an $M/M/1$ queue with $\lambda = 2$ and $\mu = 4$. Find:*
(a) P_0
(b) P_3
(c) The probability that there are more than 2 people in the system.
(d) L,L_q, W *and* W_q

Solution to Example 8 *With $\lambda = 2$ and $\mu = 4$ we have $\rho = 1/2$. Now:*
(a) $P_0 = 1 - \rho = 1 - 1/2 = 1/2$
(b) $P_3 = \rho^3 P_0 = (1/2)^3 (1/2) = 1/16$
(c) The probability that there are more than 2 people in the system.

$$\begin{aligned} 1 - (P_0 + P_1 + P_2) &= 1 - \big((1-\rho) + \rho(1-\rho) + \rho^2(1-\rho)\big) \\ &= 1 - (1/2 + 1/4 + 1/8) = 1/8 \end{aligned}$$

(d)
- $L = \frac{\rho}{1-\rho} = \frac{1/2}{1-1/2} = 1$
- $L_q = \frac{\rho^2}{1-\rho} = \frac{(1/2)^2}{1=1/2} = 1/2$
- $W = \frac{1}{\mu-\lambda} = \frac{1}{4-2} = 1/2$
- $W_q = \frac{\rho}{\mu(1-\rho)} = \frac{1/2}{4(1-1/2)} = 1/4$

A corollary of these expressions is that the queue size measures, L and L_q, rely only on the traffic intensity ρ, that is the *ratio* of the arrival and service rates, not the values themselves. Whereas the waiting time measures, W and W_q do relay on the magnitudes of λ and μ.

Example 9 *Consider two $M/M/1$ queues, one with $\lambda_1 = 6$ and $\mu_1 = 8$, and another with $\lambda_2 = 3$ and $\mu_2 = 4$. For each queue find L and W.*

Solution to Example 9 *For both queues $\rho_1 = \rho_2 = 3/4$. For the first queue:*

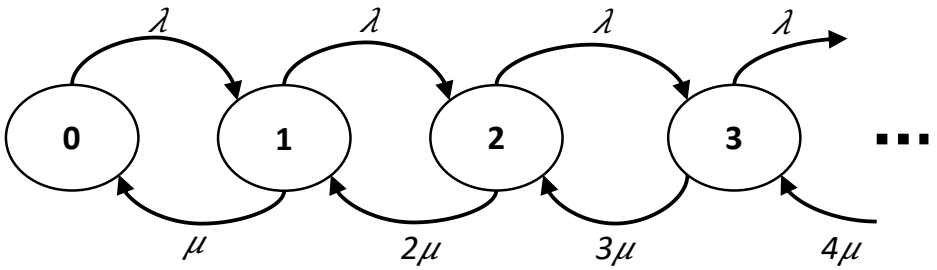
$$L = \frac{\rho_1}{1-\rho_1} = \frac{3/4}{1-3/4} = 3$$
$$W = \frac{1}{\mu_1 - \lambda_1} = \frac{1}{8-6} = 1/2$$

For the second queue:

$$L = \frac{\rho_2}{1-\rho_2} = \frac{3/4}{1-3/4} = 3$$
$$W = \frac{1}{\mu_2 - \lambda_2} = \frac{1}{4-3} = 1$$

3.6 An $M/M/\infty$ Queue

Now let's consider the case where there are infinite parallel servers. That is, there is no queue, everyone immediately enters service. The continuous-time Markov chain in this case is:



The difference now is that the service rate when in state k is $k\mu$ for all $k \in \mathbb{N}$. It does not make sense to talk about a traffic intensity now, as $\rho = \frac{\lambda}{c\mu}$ would approach 0 as $c \to \infty$. However the quantity $\theta = {}^{\lambda}/_{\mu}$ will still be useful for notation. Now we can analyse this in the exact same way as before:

$$\begin{aligned}
\lambda P_0 &= \mu P_1 \\
(\lambda + \mu) P_1 &= \lambda P_0 + 2\mu P_2 \\
(\lambda + 2\mu) P_2 &= \lambda P_1 + 3\mu P_3 \\
&\vdots \\
(\lambda + k\mu) P_k &= \lambda P_{k-1} + (k+1)\mu P_{k+1} \\
&\vdots
\end{aligned}$$

In a similar technique to above, we can guess that $P_k = \frac{\theta^k}{k!} P_0$, and we can prove this by induction:

$M/M/\infty$ Steady-State Probabilities

Theorem: An $M/M/\infty$ queue has steady-state probabilities $P_k = \frac{\theta^k}{k!} P_0$ for all $k \in \mathbb{N}$.
Proof: By induction. Base case P_1:

$$\begin{aligned}
\mu P_1 &= \lambda P_0 \\
P_1 &= \frac{\lambda}{\mu} P_0 \\
P_1 &= \theta P_0
\end{aligned}$$

Assume that $P_j = \frac{\theta^j}{j!} P_0$ for all $j \leq k$, what about P_{k+1}?

$$\begin{aligned}
\lambda P_{k-1} + (k+1)\mu P_{k+1} &= (\lambda + k\mu) P_k \\
(k+1)\mu P_{k+1} &= (\lambda + k\mu) P_k - \lambda P_{k-1} \\
(k+1)\mu P_{k+1} &= \lambda P_k + k\mu P_k - \lambda P_{k-1} \\
(k+1)\mu P_{k+1} &= \lambda \frac{\theta^k}{k!} P_0 + k\mu \frac{\theta^k}{k!} P_0 - \lambda \frac{\theta^{k-1}}{(k-1)!} P_0 \\
(k+1)\mu P_{k+1} &= P_0 \left(\lambda \frac{\theta^k}{k!} + k\mu \frac{\theta^k}{k!} - \lambda \frac{\theta^{k-1}}{(k-1)!} \right) \\
(k+1)\mu P_{k+1} &= P_0 \left(\lambda \frac{\theta^k}{k!} + \lambda \frac{\theta^{k-1}}{(k-1)!} - \lambda \frac{\theta^{k-1}}{(k-1)!} \right) \\
(k+1)\mu P_{k+1} &= \lambda \frac{\theta^k}{k!} P_0 \\
P_{k+1} &= \frac{\lambda \theta^k}{k!(k+1)\mu} P_0 \\
P_{k+1} &= \frac{\theta^{k+1}}{(k+1)!} P_0
\end{aligned}$$

which completes the proof.

How about the empty probability P_0?

$M/M/\infty$ Empty Probability

Theorem: An $M/M/\infty$ queue has empty probability $P_0 = e^{-\theta}$.
Proof: We know that $P_k = \frac{\theta^k}{k!} P_0$ for all $k \in \mathbb{N}$. We also know that the state probabilities sum to one. Therefore:

$$\begin{aligned}
\sum_{k=0}^{\infty} P_k &= 1 \\
P_0 \sum_{k=0}^{\infty} \frac{\theta^k}{k!} &= 1 \\
P_0 e^{\theta} &= 1 \\
P_0 &= \frac{1}{e^{\theta}} = e^{-\theta}
\end{aligned}$$

recognising the series expansion of the exponential function, which completes the proof.

The other performance measures are much more intuitive for an $M/M/\infty$ queue: $L_q = W_q = 0$ as no-one waits for service. $W = {}^1/_\mu$ as all the time a customer spends in the system is in service, and the average service time in ${}^1/_\mu$. Therefore from Little's law, $L = \lambda W = {}^\lambda/_\mu = \theta$. These can also be found using the steady-state probabilities above.

Now we might be able to compare $M/M/1$ queues with $M/M/\infty$ queues:

Example 10 *A call centre receives 7 calls an hour arriving in a Poisson fashion; and can process the calls at a rate of 10 per hour, where service times are Exponentially distributed. It costs* $30¢$ *an hour to keep a call on the line, whether in service or waiting. It costs* $25¢$ *an hour to hire one server. It costs* $75¢$ *an hour to run an AI robot that can process all calls simultaneously. Should the call centre hire one server, or hire the AI robot?*

Solution to Example 10 *If hiring the server, this would be an* $M/M/1$ *queue. The hourly cost* C_1 *would be:*

$$\begin{aligned} C_1 &= 30L + 25 \\ &= 30\left(\frac{\rho}{1-\rho}\right) + 25 \\ &= 30\left(\frac{{}^7/_{10}}{1-{}^7/_{10}}\right) + 25 \\ &= 30\left(\frac{7}{3}\right) + 25 \\ &= 95 \end{aligned}$$

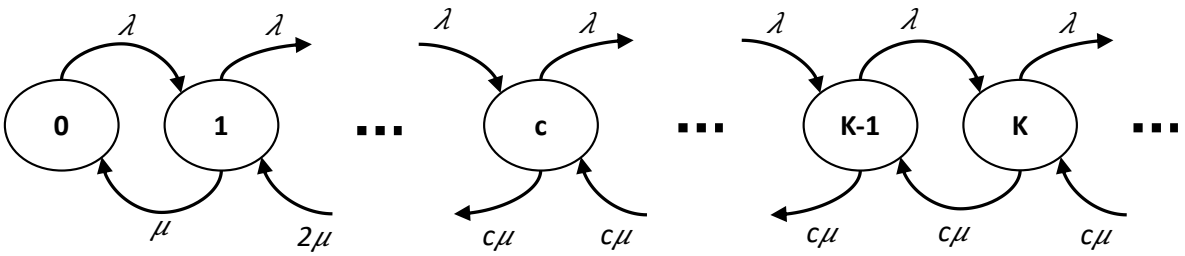
If hiring the AI robot, this would be an $M/M/\infty$ *queue. The hourly cost* C_∞ *would be:*

$$\begin{aligned} C_\infty &= 30L + 75 \\ &= 30\theta + 75 \\ &= 30\left(\frac{7}{10}\right) + 75 \\ &= 21 + 75 \\ &= 96 \end{aligned}$$

Therefore in this case it is cheaper to hire one server than the AI robot.

3.7 An $M/M/c$ Queue

Now finally let's consider an $M/M/c$ queue with infinite buffer. Our traffic intensity is $\rho = {}^{\lambda}/_{c\mu}$, and the continuous-time Markov chain would look like this:



Now we could analyse this as before. However, we can think a little more heuristically. There are two regions of this Markov chain that we can analyse separately:

- for states k where $k < c$, then no customer queues. This region behaves similarly to the $M/M/\infty$ case. That is:

$$\begin{aligned} P_k &= \frac{\theta^k}{k!} P_0 \\ &= \frac{(c\rho)^k}{k!} P_0 \end{aligned}$$

- for states k where $k \geq c$, then newly arriving customers join a queue, and from this point on there is an effective service rate of $c\mu$. That is, the denominator $k!$ in the above expression becomes $c!c^{k-c}$:

$$\begin{aligned} P_k &= \frac{\theta^k}{c!c^{k-c}} P_0 \\ &= \frac{(c\rho)^k}{c!c^{k-c}} P_0 \\ &= \frac{c^k \rho^k}{c!c^{k-c}} P_0 \\ &= \frac{c^c \rho^k}{c!} P_0 \end{aligned}$$

- and finally:

$$\begin{aligned}
\sum_{k=0}^{\infty} P_k &= 1 \\
\sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} P_0 + \sum_{k=c}^{\infty} \frac{c^c \rho^k}{c!} P_0 &= 1 \\
P_0 \left(\sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} + \sum_{k=c}^{\infty} \frac{c^c \rho^k}{c!} \right) &= 1 \\
P_0 \left(\frac{(c\rho)^c}{c!(1-\rho)} + \sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} \right) &= 1 \\
P_0 &= \frac{1}{\left(\frac{(c\rho)^c}{c!(1-\rho)} + \sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} \right)}
\end{aligned}$$

$M/M/c$ Expected Queue Size

Theorem: For an $M/M/c$ queue we have:

$$L_q = \frac{\rho^{c+1} c^c}{c!(1-\rho)^2} P_0$$

Proof: We only need to consider the states where there is a customer queueing:

$$\begin{aligned}
L_q &= \sum_{k=c}^{\infty} (k-c) P_k \\
&= \sum_{k=c}^{\infty} (k-c) \frac{c^c \rho^k}{c!} P_0 \\
&= \sum_{k=0}^{\infty} k \frac{c^c \rho^{k+c}}{c!} P_0 \\
&= \frac{c^c \rho^c}{c!} P_0 \sum_{k=0}^{\infty} k \rho^k \\
&= \frac{c^c \rho^{c+1}}{c!(1-\rho)^2} P_0
\end{aligned}$$

Now all other measures, L, W, and W_q can be found using Little's law.

Chapter 4

Stochastic Simulation

Learning Outcomes:

- Be able to understand stochastic simulation and the modelling process;
- Be able to estimate definite integrals and unknown probability distributions using Monte Carlo simulation;
- Be able to simulate queueing systems using discrete event simulation;
- Be able to recognise different simulation software.

4.1 Introduction

We have so far seen a number of different probabilistic scenarios that we can analyse mathematically. These problems have been of specific forms that are amenable to be analysed with the tools we have learned, however not all problems are tractable. Simulation is one way to overcome this, by considering the frequency approach to probability. That is: if a probability experiment could be conducted an infinite number of times, then the true results could be obtained, without needing to analyse the system. I.e. if I toss a coin many times, I would expect around half the tosses to come up heads.

Performing a probability experiment a number of times may be costly, in terms of both value and time, or even impossible. For example if the probability experiment is something like: what is the probability of a patient arriving to A&E dying before the end of the day when there are only 2 doctors on duty? It is costly in terms of lives, it is costly in terms of days, costly in terms of money, and may be an impossible experiment to run due to ethics. Therefore we can use simulation, which involves recreating this using a computer in a risk-free environment.

4.2 Laws of Large Numbers

The laws of large numbers give an intuitive reason why and how simulation works. These laws will not be formally stated or proven here, however an understanding and interpretation of them will be vital.

- Let X be a random variable with:
  - probability distribution F
  - mean μ
  - variance σ^2
  - and probability of being less than x of p_x.
- Let $\{X_1, X_2, \ldots, X_n\}$ be a random sample of size n of the random variable X, with:
  - empirical probability distribution F_n
  - mean $\overline{X}_n$
  - variance $\text{Var}(X_n)$
  - and number of observations less than x of N_x.

Now we have three useful results (not stated here in a rigorous manner):

The Strong Law of Large Numbers

$$\overline{X}_n \to \mu \quad \text{as} \quad n \to \infty \tag{4.1}$$

$$\text{Var}(X_n) \to \sigma^2 \quad \text{as} \quad n \to \infty \tag{4.2}$$

Borel's Law of Large Numbers

$$N_x \to p \quad \text{as} \quad n \to \infty \tag{4.3}$$

Glivenko–Cantelli Theorem

$$F_n \to F \quad \text{as} \quad n \to \infty \tag{4.4}$$

That is, the larger the sample, the sample means approach the true means; sample variances approach true variances; proportions approach the true probabilities; and empirical distributions approach the true distributions.

4.3 Sampling Pseudo-Random Numbers

True randomness cannot be created on a computer, as computers are deterministic. Instead, a computer must Uniformly distributed generate *pseudo-random numbers*, numbers that are generated by a deterministic function, however without knowing that function act and behave like Uniformly randomly sampled numbers. That is, for a large enough sequence $X_0, X_1, \ldots X_n$, they exhibit:

- **uniformity**: $\mathbb{P}(X_i < x) = x$ for all $x \in [0, 1]$.
- **independence**: $\mathbb{P}(X_0 < x_0, X_1 < x_1, \ldots X_n < x_n) = \prod_{i=0}^{n} \mathbb{P}(X_i < x_i)$

For most uses we also require **high-dimensional equidistribution**, that is if we create high-dimensional points from consecutive tuples of numbers in the sequence, the sequence of high-dimensional points fills the volume of the high dimensional hypercube by is filled uniformly as the sequence progresses.

Another feature of sequences of pseudo-random number is that the function used to generate them is deterministic, and can be **seeded**. That is, with a particular seed, or starting value, we will always produce the exact same sequence of pseudo-random numbers, and we will produce a different sequence of pseudo-random numbers with a different seed. This ensures any experiments carried out with the pseudo-random numbers are **reproducible**.

One early way of producing pseudo-random numbers was called a linear congruential generator. It is parametrised by a seed X_0, a multiplier A, a constant C, and a modulus M such that:

$$X_{n+1} = (AX_n + C) \mod M$$

and clever choices for parameters A, C and M can ensure uniformity and independence, e.g. if M is a large power of 2, $M > A, C$, M and C are co-prime, and $A - 1$ is divisible by all prime factors of M. This generally produces a sequence of random integers between 0 and M, and dividing by M gives the required $X \sim U(0, 1)$.

An example is given in the table below, with $M = 8$, $A = 5$, $C = 3$ and $X_0 = 7$:

n	X_n	X_n/M
0	7	0.875
1	6	0.750
2	1	0.125
3	0	0.000
4	3	0.375
5	2	0.250
6	5	0.625
7	4	0.500
8	7	0.875
9	6	0.750
10	1	0.125
11	0	0.000
12	3	0.367
	⋮	

We see that every integer between 0 and M occurs, in some deterministic but undetectable order, and then the sequence repeats. Therefore, choosing M large enough can ensure a sufficiently long non-repeating stream of pseudo-random numbers. E.g. $M = 2^{32}$.

Modern programming languages come with very sophisticated pseudo-random number generators built-in. For example Python uses an algorithm called the Mersenne Twister:

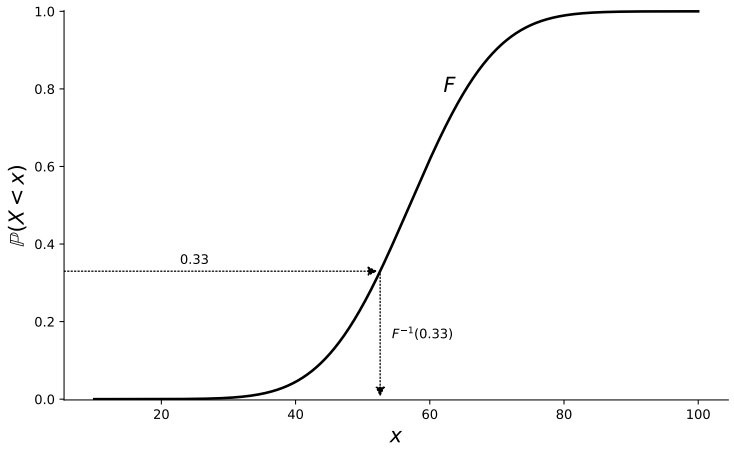
```
>>> import random
>>> random.seed(7777)
>>> random.random()
0.8170477907294282
>>> random.random()
0.5311536664127009

>>> random.seed(7777)
>>> random.random()
0.8170477907294282
>>> random.random()
0.5311536664127009
```

Inverse Distribution Method

We have seen how to "sample" pseudo-random numbers that are Uniformly distributed between 0 and 1. It is useful to be able to sample pseudo-random numbers from other probability dis-

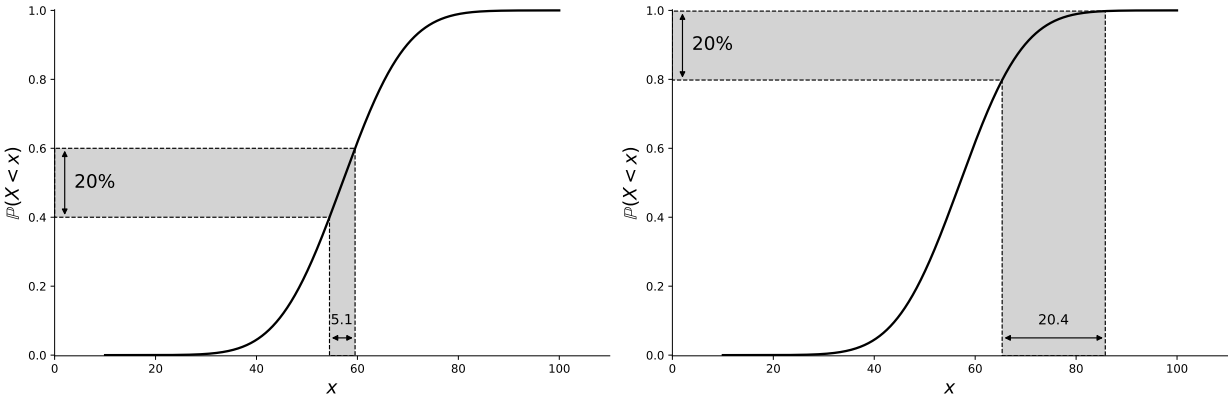
tributions. One way to do this is to *transform* the Uniformly distributed variable into another. Consider the CDF, F, below:



The y-axis, that is the $\mathbb{P}(X < x)$ is a probability, and is bounded by 0 below and 1 above. The **uniformity** property above implies that sampling a random number less than $x \in [0, 1]$ occurs with probability x, which corresponds exactly to this y-axis. I.e. Uniformly distributed pseudo-random numbers between 0 and 1 correspond to sampling a probability.

We wish to sample from the x-axis, and so we can use the inverse CDF, F^{-1} to transform the sampled probability into a sampled value. E.g., If we have sample the probability 0.33, then we can transform this into the value $F^{-1}(0.33)$.

A common misconception here would be that as F is bijective, then and all points in the range are equally likely, then all points in the domain must also be equally likely. This is not true, and can be shown visually by considering sampling from intervals. The image below shows two equally sized intervals in the range, but they map to two different sized intervals in the domain. Therefore the probability of sampling a number lying in both intervals are equal, 20%, but the width of the two intervals are different, showing that some intervals are more likely to be sampled than others.



Example 11 *A pseudo-random number generator gives the following five Uniformly distributed numbers: (0.84442, 0.75795, 0.42057, 0.25892, 0.51127). Transform these into five Exponentially distributed numbers with* $\lambda = {}^{1}/_{15}$.

Solution to Example 11 *For the Exponential distribution,* $F(x) = 1 - e^{-\lambda x}$, *and we want* $F^{-1}(x)$:

$$\begin{aligned} y &= 1 - e^{-\lambda z} \\ e^{-\lambda z} &= 1 - y \\ -\lambda z &= \ln(1-y) \\ z &= -\frac{1}{\lambda}\ln(1-y) \\ z &= \frac{1}{\lambda}\ln\left(\frac{1}{1-y}\right) \end{aligned}$$

And so $F^{-1}(x) = \frac{1}{\lambda}\ln\left(\frac{1}{1-x}\right)$.
We can now apply this to each of the five Uniformly distributed random number to get five Exponentially distributed random numbers with rate ${}^{1}/_{15}$:

$$\begin{aligned} F^{-1}(0.84442) &= 27.90893 \\ F^{-1}(0.75795) &= 21.27916 \\ F^{-1}(0.42057) &= 8.18566 \\ F^{-1}(0.25892) &= 4.49470 \\ F^{-1}(0.51127) &= 10.73918 \end{aligned}$$

This can apply to any probability distribution we can think of:

Example 12 *A pseudo-random number generator gives the following five Uniformly distributed numbers: (0.84442, 0.75795, 0.42057, 0.25892, 0.51127). Transform these into five numbers sampled from the following sinusoidal PDF:* $f(x) = \frac{1}{2}\sin(x)$.

Solution to Example 12 *First find the CDF:*

$$\begin{aligned} F(x) &= \int_0^x f(t)dt \\ &= \frac{1}{2}\int_0^x \sin(t)dt \\ &= \frac{1}{2}\left[-\cos(t)\right]_0^x \\ &= \frac{1}{2}\left(-\cos(x)+\cos(0)\right) \\ &= \frac{1}{2}\left(1-\cos(x)\right) \end{aligned}$$

Then find the inverse CDF:

$$\begin{aligned} y &= -\frac{1}{2}\left(1-\cos(z)\right) \\ 2y &= 1-\cos(z) \\ 2y-1 &= \cos(z) \\ \cos^{-1}(2y-1) &= x \end{aligned}$$

And so $F^{-1}(x) = \cos^{-1}(2x-1)$.
We can now apply this to each of the five Uniformly distributed random number to get:

$$\begin{aligned} F^{-1}(0.84442) &= 0.81091 \\ F^{-1}(0.75795) &= 1.02874 \\ F^{-1}(0.42057) &= 1.73033 \\ F^{-1}(0.25892) &= 2.07392 \\ F^{-1}(0.51127) &= 1.54825 \end{aligned}$$

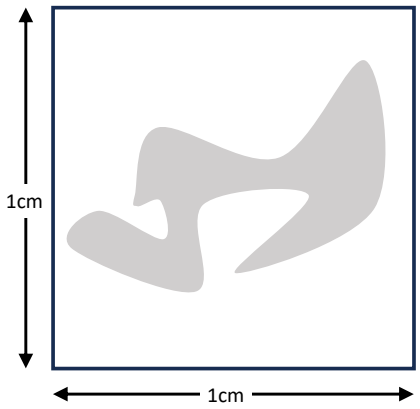
4.4 Monte Carlo Simulation

Monte Carlo simulation is a broad term than involves repeatedly randomly sampling from probability distributions in order to obtain numerical results. We can solve deterministic and stochastic problems. Note that in this section, due to the need to sample a large amount of random numbers, examples will take place in code.

Deterministic Problems

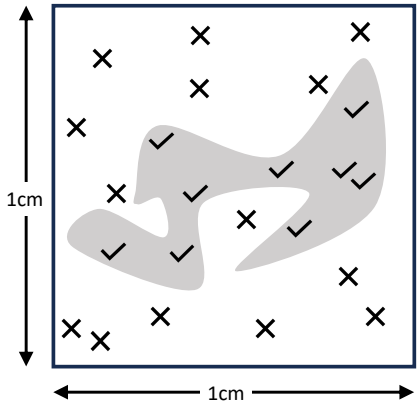
Using randomness to solve deterministic problems might seem unintuitive, however we will look at a useful technique that conflates the idea of a probability into a geometric area.

Consider the image below:



We have a 1cm by 1cm frame, and a smudge in the middle of the frame. What is the area of the smudge?

Now assume that we can randomly sample points inside the frame, uniformly in space, and that it is easy to check whether a point lies inside the smudge or not:

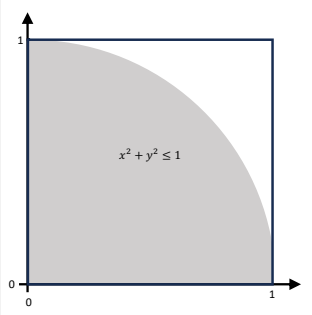


then, as the number of points sampled increases: the proportion of points inside the smudge will approach the proportion of the area the smudge takes up inside the frame. Then knowing the area of the frame gives the area of the smudge.

The assumption that we can randomly sample points inside the frame is satisfied by properties of pseudo-random numbers: Let (X, Y) be a random point in space, where $X \sim U(0, 1)$ and $Y \sim U(0, 1)$. The assumption that we can easily check if a point lies within the area of interest depends on the problem itself.

We will look at two examples, estimating π, and evaluating an integral.

Example 13 *Consider the quarter circle defined in the first quadrant by* $x^2 + y^2 \leq 1$*:*



This is a quarter of a circle with radius 1*, and so we know its area is given by* $\frac{1}{4}\pi$*. Find a good estimate of the value of* π *by Monte Carlo simulation, using* $N = 10$*,* $N = 1,000$ *and* $N = 100,000$ *random points.*

Solution to Example 13 *We have:*

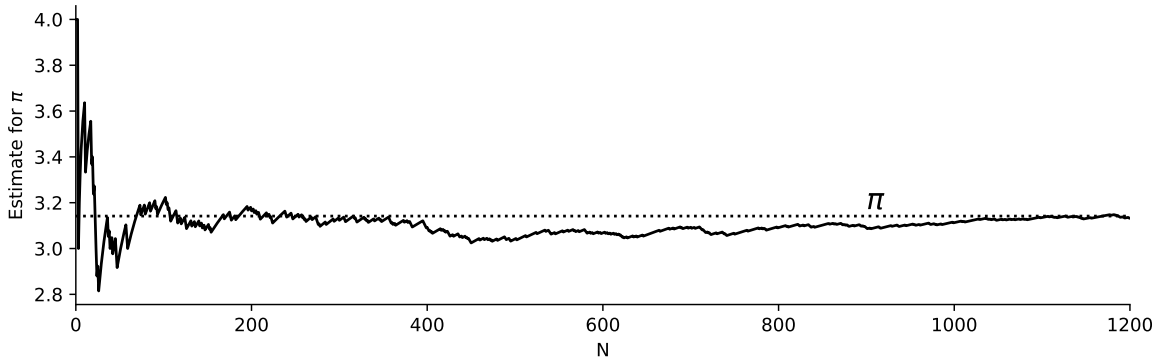
$$\frac{\textit{Area of quarter circle}}{\textit{Area of unit square}} = \frac{\textit{Number of points in circle}}{\textit{Number of points in square}}$$

The left hand side is equal to $\frac{1}{4}\pi$*, and the right hand side is found using Monte Carlo simulation:*

```
>>> import random
>>> random.seed(77)
>>> def estimate_pi(N):
...     number_in_circle = 0
...     for point in range(N):
...         x = random.random()
...         y = random.random()
...         if (x ** 2) + (y ** 2) <= 1:
...             number_in_circle += 1
...     pi = (number_in_circle / N) * 4
...     return pi

>>> estimate_pi(10)
4.0
>>> estimate_pi(1000)
3.288
>>> estimate_pi(100000)
3.13692
```

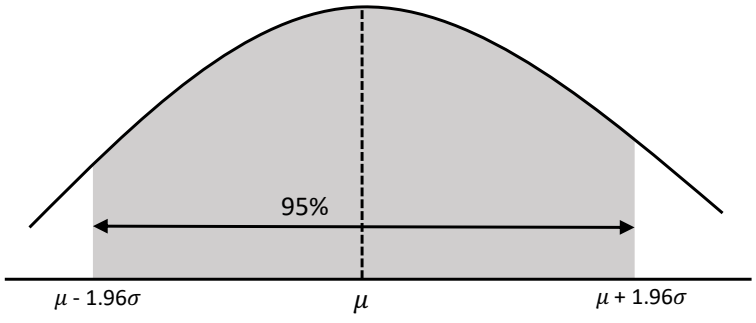
In that example we could see how increasing the number of samples improves our estimate of π. This is visualised below, as the sample size N increases, the estimate gets closer and closer to the true value:



Accuracy of Estimates

In Monte Carlo simulation we have N trials and record the amount of successes, n. This gives $\hat{p} = {}^{n}/_{N}$, an estimate for some true proportion p. The probability of observing n is given by the binomial distribution. When $Np > 5$ and $N(1-p) > 5$, that is at least five successes and at least five failures, this can be approximated by the Normal distribution, $n \sim N(\mu = Np, \sigma = \sqrt{Np(1-p)})$.

Recall that a Normally distributed variable with mean μ and standard deviation σ has 95% chance of being in the interval $(\mu - 1.96\sigma, \mu + 1.96\sigma)$:



Therefore with $\mu = Np$ and $\sigma = \sqrt{Np(1-p)}$:

Monte Carlo Estimate Confidence Interval

In a Monte Carlo simulation estimating a proportion p, with N trials and n successes, then $\hat{p} = {}^{n}/_{N}$ is an estimate of p with confidence interval:

$$CI = \left(\hat{p} - 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{N}},\ \hat{p} + 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{N}} \right)$$

that is, if we were to repeat the experiment many times, then the true value p would fall inside the interval CI 95% of the time.

We can see that as N increases the confidence interval narrows, and we become more confident in our estimate. Note that this is only true if we can use the Normal approximation, that is when $Np > 5$ and $N(1-p) > 5$. Generally, if we can construct our experiment such that $p \approx {}^{1}/_{2}$, then we would need at least 10 trials for the Normal approximation to be a good approximation.

Note that the value 1.96 comes from the inverse CDF of the Normal distribution. Other values can be used, e.g. for a 90% confidence interval use 1.64, and for a 99% confidence interval use 2.58.

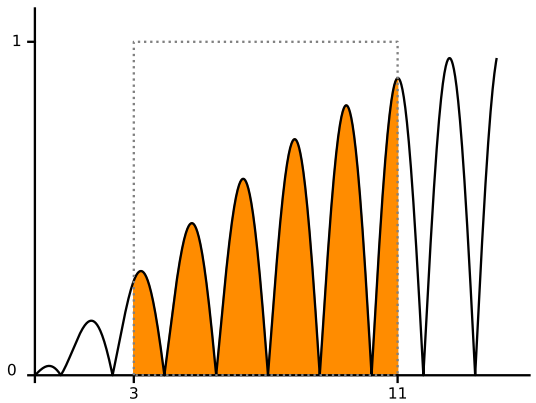
Let's do another example, estimating the value of a definite integral.

Example 14 *Estimate the following integral using* 500 *trials of a Monte Carlo simulation:*

$$I = \int_{3}^{11} \left| \sin\left(\frac{x}{10}\right) \cos\left(2x\right) \right| dx$$

Give a 95% confidence interval for your estimate.

This is a difficult integral to solve analytically, so Monte Carlo simulation is useful here. The integrand is shown below, and we wish to estimate the orange area under the curve:



Checking if a point (x, y) is under the curve or not is easy, just compare $y < f(x)$ where f is the integrand. We now need to sample points in the indicated rectangle of area 8.

Solution to Example 14 *First, the integrand:*

```
>>> import math
>>> def integrand(x):
...     return abs(math.sin(x / 10) * math.cos(2 * x))
```

Now sample 500 points:

```
>>> N = 500
>>> n = 0
>>> random.seed(0)
>>> for trial in range(N)
...     point = (3 + (random.random() * 8), random.random())
...     if point[1] <= integrand(point[0]):
...         n += 1
>>> p = n / N
>>> estimated_value_of_integral = p * 8
>>> estimated_value_of_integral
3.44
```

Now for a confidence interval:

```
>>> lower = p - 1.96 * ((p * (1 - p) / N) ** 0.5)
>>> upper = p + 1.96 * ((p * (1 - p) / N) ** 0.5)
>>> (lower * 8, upper * 8)
(3.0928375762269193, 3.7871624237730805)
```

and so we can be 95% confident that the true value of the definite integral I lies in the interval $(3.093, 3.787)$.

Stochastic Problems

Another use of Monte Carlo simulation is to consider stochastic scenarios. This is useful in cases where we might not be able to analyse some random variables analytically. We could break this random variable down into component random variables that we are able to sample (for example via the inverse distribution method), and combine them accordingly.

Example 15 *My drive two work consists of driving down two roads, and there is a possibility of needing to stop and wait for traffic at the intersection:*

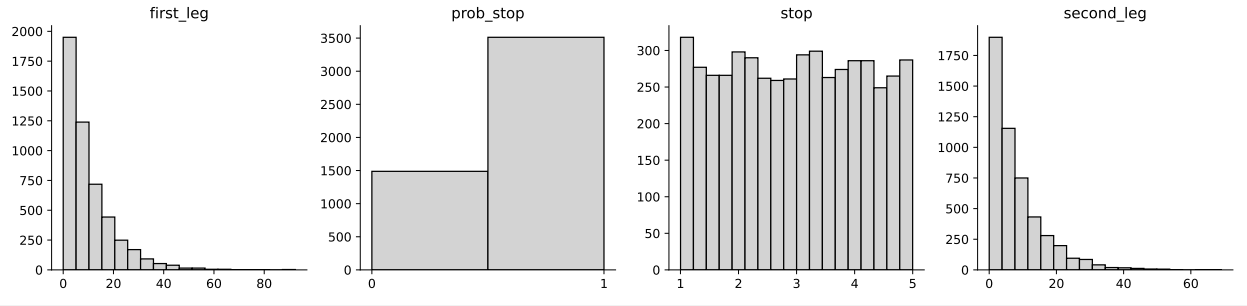
- *the time taken to drive first leg in minutes is Exponentially distributed with rate* $1/10$*;*
- *the probability of needing to stop for traffic at the intersection is 0.7;*
- *if stopping is required, the amount of time waiting for traffic is Uniformly distributed between 1 and 5 minutes;*
- *the time taken to drive the second leg in minutes is Exponentially distributed with rate* $1/8$*.*

What is the distribution of the total journey time? What is the mean journey time? What is the probability that the journey will take more than 30 minutes?

Solution to Example 15 *Each component part can be sampled, lets choose* $N = 5000$ *trials:*

```
>>> import random
>>> random.seed(0)
>>> N = 5000
>>> first_leg = [random.expovariate(1/10) for _ in range(N)]
>>> prob_stop = [1 if random.random() < 0.7 else 0 for _ in range(N)]
>>> stop = [1 + (4 * random.random()) for _ in range(N)]
>>> second_leg = [random.expovariate(1/8) for _ in range(N)]
```

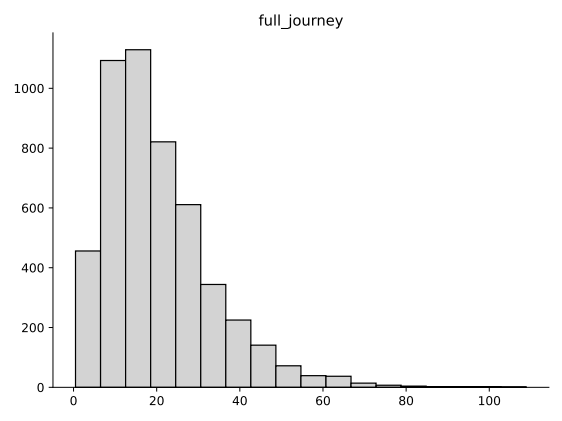
We can plot histograms of these to visualise the component parts:



We can combine them in the appropriate way:

```
>>> full_journey = [
...     x1 + (p * s) + x2 for x1, p, s, x2 in zip(
...         first_leg, prob_stop, stop, second_leg
...     )
... ]
```

Solution to Example 15 (continuing from p. 62) *This combined distribution looks like:*



And we can find the mean, and probability of being under 30 minutes:

```
>>> mean = sum(full_journey) / N
>>> mean
20.35993088090077
>>> prob_under_30 = sum(x > 30 for x in full_journey) / N
>>> prob_under_30
0.1872
```

4.5 Discrete Event Simulation

Discrete Event Simulation, or DES, is a complex Monte Carlo simulation, usually of a queueing system. Generally when building and running a DES, sampling component random variables and their logical combinations are taken care of for us by some *simulation software*. Their combination can be quite complex, depending on the system being simulated.

For example, consider an $M/M/1$ queue, and we are interested in the distribution of waiting times of the customers. The random variables to sample are the inter-arrival times of the customers, and their service times. The relevant combinations required to find the waiting times are shown in the table below:

Customer i	Sampled inter-arrivals I_i	Arrival date $A_i = \sum_{k=0}^{i} A_i$	Service start date $S_i = \max(A_i, E_{i-1})$	Waiting time $W_i = S_i - A_i$	Sampled service time Z_i	Service end date $E_i = S_i + Z_i$
1	1.657	1.657	1.657	0.000	0.807	2.463
2	2.745	4.402	4.402	0.000	0.979	5.381
3	0.280	4.682	5.381	0.699	0.261	5.642
4	2.721	7.403	7.403	0.000	2.012	9.415
5	0.146	7.549	9.415	1.866	0.596	10.011
6	0.075	7.624	10.011	2.387	1.358	11.369
7	1.243	8.867	11.369	2.502	2.329	13.698
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Try to imagine what the logical combinations of random variables would be like for an $M/M/c$ queue, or for something more complex such as networks of queues, where after completion in one queue customers join another. It gets difficult! Therefore, simulation software allows us to focus on the *model building* rather than the logical implementation.

There are many different software available for conducting DES, and simulation in general. Some of these are open-source, and some are commercial software; some are code based, while others are visual and are interacted with via a Graphical User Interface. These include:

Open-source:

- Ciw (in Python)
- SimPy (in Python)
- Simmer (in R)
- NetLogo

Commercial:

- SIMUL8
- Anylogic
- Arena
- FlexSim

The main software we will use here is Ciw, a Python library for conducting discrete event simulations of queueing networks. It will need to be installed on your computer via `pip install ciw`, and full documentation, including tutorials, are given at `https://ciw.readthedocs.io`.

The code that builds a simulation of an $M/M/1$ queue, with arrivals $\lambda = 10$ per hour, and services $\mu = 15$ per hour, is:

```
>>> import ciw
>>> N = ciw.create_network(
...     arrival_distributions=[ciw.dists.Exponential(rate=10)],
...     service_distributions=[ciw.dists.Exponential(rate=15)],
...     number_of_servers=[1]
... )
```

Now we can run the simulation, say for 50 time units:

```
>>> ciw.seed(0)
>>> Q = ciw.Simulation(N)
>>> Q.simulate_until_max_time(50)
```

And we can collect data describing everything that happened during the simulation run. We can then observe some statistic of interest, for example the average waiting time:

```
>>> recs = Q.get_all_records()
>>> waits = [r.waiting_time for r in recs]
>>> average_wait = sum(waits) / len(waits)
>>> average_wait
0.0957478641425804
```

This number, `0.0957478641425804`, isn't a good estimate of the true average waiting time. Compare it to the formula we used in Chapter 3:

$$W_q = \frac{\rho}{\mu(1-\rho)} = \frac{{}^{10}/_{15}}{15 \times (1 - {}^{10}/_{15})} = \frac{2}{15} = 0.1\dot{3}$$

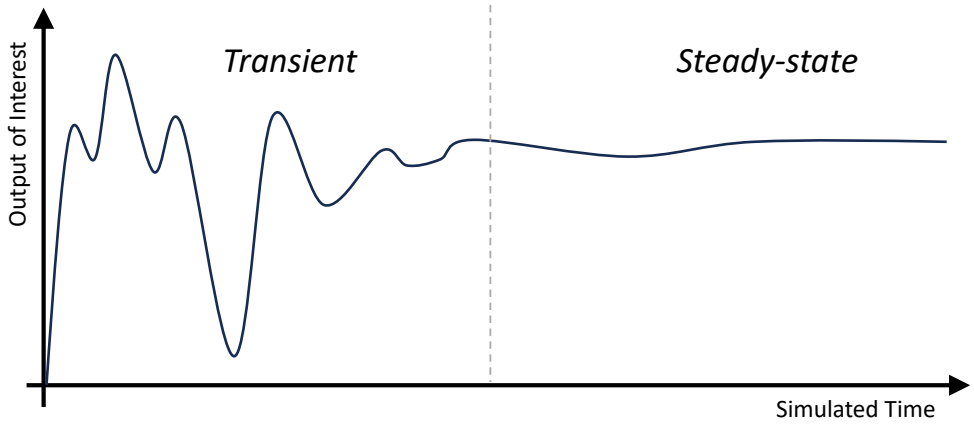
The discrepancy here is due to two reasons: transient behaviour, and number of iterations.

Transient Behaviour

Generally there are two types of model:

- **Terminating**: where there is a defined opening and closing time, for example, a simulation of a bank. The bank opens each day empty, and closes each evening, resetting its state. In these cases the transient behaviour is important, that is how the system progresses over time.
- **Steady-state**: where there is no defined start time or end time, and system runs for a very long period of time, for example a simulation of a 24 hour production line. In these cases we generally case about the steady-state of the system.

Unfortunately, without knowing what the steady-state is, we cannot begin the system in steady-state. Therefore simulations must begin in some arbitrary state, generally the empty-state, that is, with no customers present. In this state, all newly arriving customers will have no waiting time, biasing the result. This means between the arbitrary starting state and steady-state, the system will exist in transient states:



To overcome this bias we utilise a **warm-up time**. This is a period of time at the beginning of the simulation run where results aren't collected. We can utilise a warm-up time in Ciw by filtering results:

```
>>> warmup_time = 25
>>> waits = [
...     r.waiting_time for r in recs if r.arrival_date > warmup_time
... ]
```

We want to choose is large enough so that observations in the transient states are not collected. A simple way to choose this is by calculating a moving average and looking at the plot.

Number of Iterations

We saw in Section 4.4 on Monte Carlo simulation that as the number of observations increases, we get a better estimate for the statistic we are measuring. This is true for DES, but what does number of observations mean here?

- For **terminating** models one run of the simulation is one observation. Therefore we need to re-run the simulation many times, with different random seeds. This is called running *trials*. When using Ciw, we would do this by using for-loops.
- For **steady-state** models, we can also run trials, but we can also simply run the simulation for a very long time, as we gather more and more observations.

Some examples:

Example 16 *Estimate the average waiting time for a $G/G/3$ queue, where the inter-arrival times are Uniformly distributed between 4 and 12 minutes, and the service times are distributed according to a Gamma distribution shape parameter 2.5 and scale parameter 7.*

Solution to Example 16 *This is a steady-state model, so we will conduct one long simulation (seven days) run with a warm-up time (one day). Time units are in minutes:*

```
>>> import ciw
>>> max_simulation_time = 60 * 24 * 8
>>> warmup = 60 * 24
>>> N = ciw.create_network(
...     arrival_distributions=[ciw.dists.Uniform(lower=4, upper=12)],
...     service_distributions=[ciw.dists.Gamma(shape=2.5, scale=7)],
...     number_of_servers=[3]
... )
>>> ciw.seed(0)
>>> Q = ciw.Simulation(N)
>>> Q.simulate_until_max_time(max_simulation_time)
>>> recs = Q.get_all_records()
>>> waits = [r.waiting_time for r in recs if r.arrival_date > warmup]
>>> sum(waits) / len(waits)
1.8018106484544456
```

And a terminating example:

Example 17 *A bicycle repair workshop is open for 9 hours a day:*

- *bikes arrive randomly at a rate of 15 per hour,*
- *they first queue for an inspection, which can complete at a rate of 20 per hour,*
- *20% of the bikes do not need repairing, and leave the workshop,*
- *80% of bikes then join a queue for repair, which can complete at a rate of 10 per hour.*

The workshop currently has one employee inspecting the bikes, and one employee repairing the bikes. They care about the proportion of bikes that spend longer than 0.5 hour in total at the repair shop. They want to employ an extra person to reduce this proportion, would it be more beneficial for them if the new employee inspected the bikes or repaired the bikes?

Solution to Example 17 *This is a terminating model, and so we need to repeat the experiment many times. We can do this with a function:*

```
>>> import ciw
>>> import pandas as pd
>>> def get_proportion(n_inspectors, n_repairers):
...     N = ciw.create_network(
...         arrival_distributions=[
...             ciw.dists.Exponential(rate=15),
...             None
...         ],
...         service_distributions=[
...             ciw.dists.Exponential(rate=20),
...             ciw.dists.Exponential(rate=10)
...         ],
...         number_of_servers=[n_inspectors, n_repairers],
...         routing=[[0.0, 0.8], [0.0, 0.0]]
...     )
...     Q = ciw.Simulation(N)
...     Q.simulate_until_max_time(9)
...     recs = Q.get_all_records()
...     R = pd.DataFrame(recs).groupby('id_number')
...     time_in_system = R['exit_date'].max() - R['arrival_date'].min()
...     return (time_in_system > 0.5).mean()
```

We will also write a function to run many trials of this:

```
>>> def run_trials(n_inspectors, n_repairers, trials):
...     ps = []
...     for trial in range(trials):
...         ciw.seed(trial)
...         p = get_proportion(n_inspectors, n_repairers)
...         ps.append(p)
...     return sum(ps) / trials
```

Now we can run the original scenario:

```
>>> run_trials(n_inspectors=1, n_repairers=1, trials=20)
0.5339246569473143
```

Solution to Example 17 (continuing from p. 68) *and the two new scenarios:*

```
>>> run_trials(n_inspectors=2, n_repairers=1, trials=20)
0.44444581769225433
>>> run_trials(n_inspectors=1, n_repairers=2, trials=20)
0.19943800806935968
```

We see that both scenarios improve the proportion staying over 0.5 hours, however hiring an extra employee to repair the bikes would be much more beneficial than hiring an extra employee to inspect the bikes.

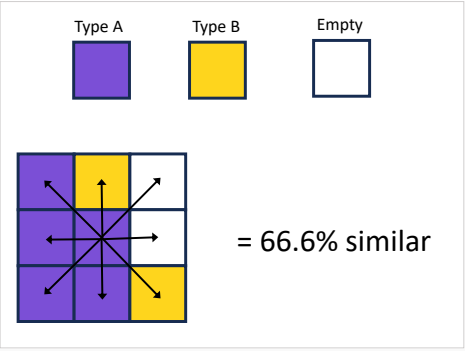
Note that this example modelled a network of queues. This is a staple of DES modelling.

4.6 Agent-Based Modelling

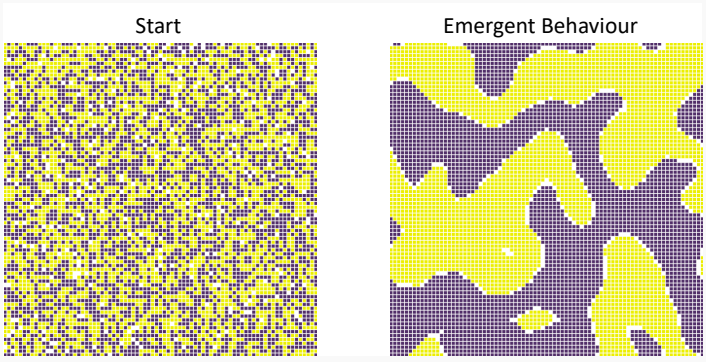
This section is given for interest, as an idea another type of simulation modelling called *agent-based-modelling*. These models generally consist of many entities or agents following their own rules, or *micromotives*, and as the model runs we observe the whole system's emergent behaviour, or *macrobehaviour*. Here are some early noteworthy models to demonstrate the general ideas:

Schelling's Model of Segregation

In this model there is an $N \times N$ a grid of properties, representing a city. Agents are households that can solely occupy a property on the grid. These households can be of two types, A and B. There are less agents that properties, so a certain percentage remain empty. A household is "happy" when the percentage of their immediate neighbours (those directly above, below, to the right, left and diagonally) who are of the same type as them is above some threshold p.



At each clock tick, any unhappy households move to a randomly chosen empty property. This continuous until all households are happy. We see than the households naturally segregate themselves:



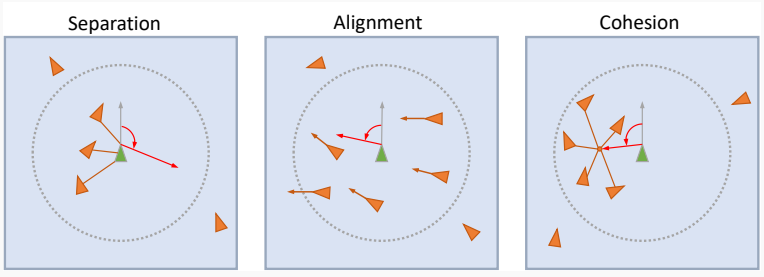
In the example above, the similarity threshold was 50%, while in the emergent behaviour the average neighbour similarity is 96%. Generally this model shows that even a small preference for similar neighbours can result in an extremely segregated society.

BOIDS

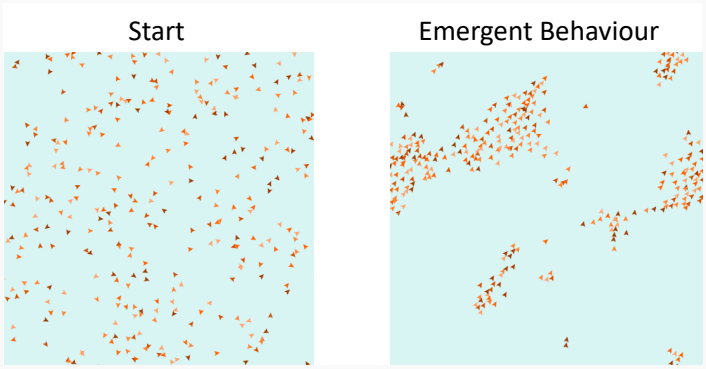
In this model the world is populated by virtual birds, initially scattered randomly, and all facing random angles. At each clock tick, the virtual birds take a fraction of a step forward in the direction they are facing, and then turn a fraction of an angle, according to three rules:

i *separation*: turn to avoid crowding nearby birds,
ii *alignment*: turn towards the average direction nearby birds are heading,
iii *cohesion*: turn towards the average position of nearby birds.

Here 'nearby birds' mean those within a given radius of the bird's position.



We see that eventually natural bird flocking behaviour emerges:



This seems to hint that real life flocking birds do so by following their own, individual simple rules, rather than controlled by a central leader.

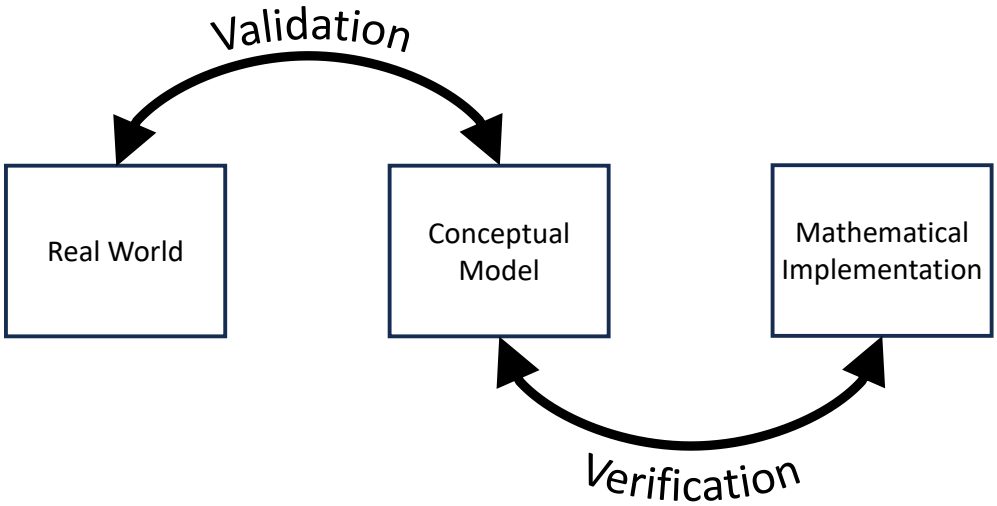
4.7 Modelling Process

Simulation modelling, and mathematical modelling more generally, is more than just the mathematics behind it. There is a whole process that modellers go through to ensure that the models are good. Some important concepts will be introduced here.

Modelling Concepts

- **Conceptual model**: This is a simplified, abstract representation of the real-world system being modelled. In contains the key details that are of interest, and ignores details that we deem unimportant for the model.
- **Validation**: This is the process of ensuring that the conceptual model truly does represent the real-world system being modelled.
- **Verification**: This is the process of ensuring that the mathematical methodology used correctly implements the conceptual model.

That is, validation ensures that you are solving the correct problem. Verification ensures that you are solving the problem correctly.



There are a number of techniques we can use to validate and verify our models. For example:

Validation

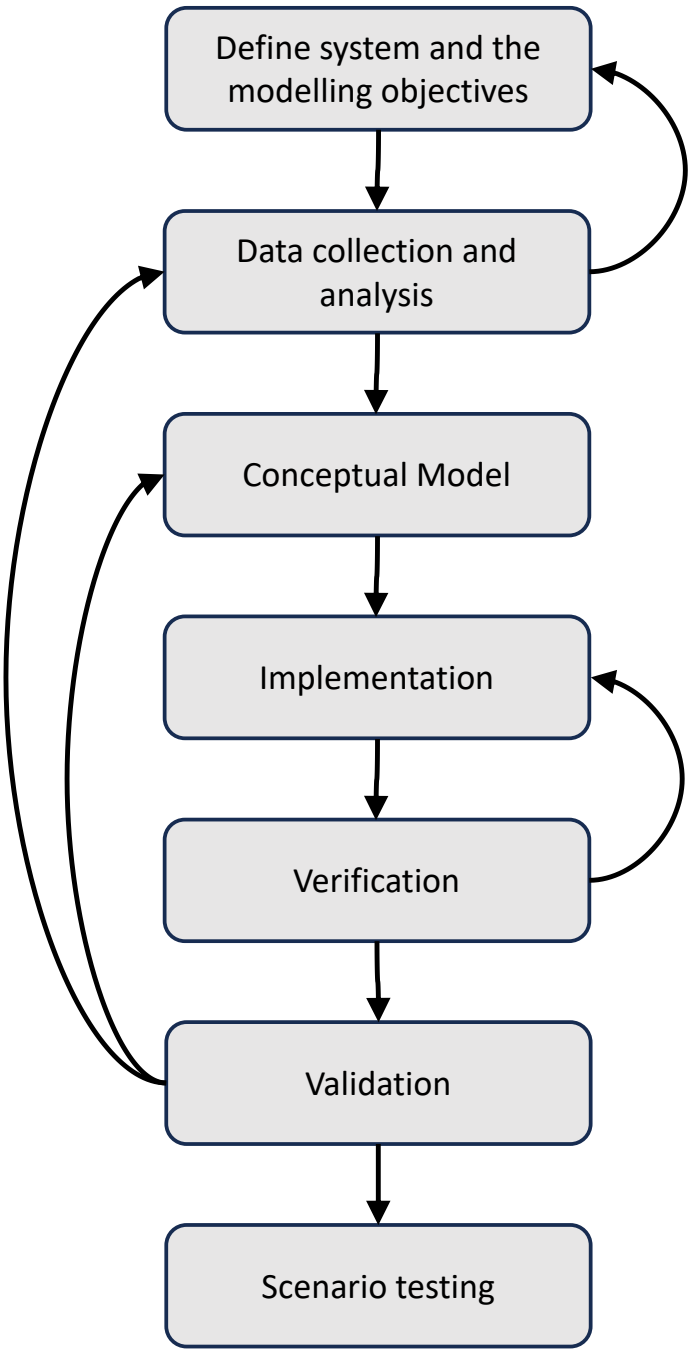
- face validation: asking experts in the real system to assess the accuracy of the conceptual model,
- considering model assumptions: listing and checking all assumptions made in the conceptual model, e.g. assumptions about probability distributions used,
- output data: comparing the output data of the correctly implemented mathematical model with data collected in the real world.

Verification

- tracing: following individual entities throughout the simulation run to test the model logic,
- comparing to analytical results: if some components of the model can be modelled analytically (e.g. Markovian queues), compare the output of the simulated component to the analytical model,
- extreme tests: change some input parameters to some extreme values, and see if the output makes sense, e.g. flooding the system with a ridiculous amount of arrivals should increase waiting times.

There is a large overlap between simulation model verification and comprehensive automated testing of software, and many of the same steps and techniques are used.

When building a model, both conceptual and mathematical, we don't just dive right in. There are steps to take. First we define the objectives, this will help with the conceptual model, and deciding which simplifications and assumptions we can make. This is iterated upon using validation and verification. Once the model is built, only then can what-if scenarios be tested. This is summarised in the following flow chart:


Define system and the modelling objectives
Data collection and analysis
Conceptual Model
Implementation
Verification
Validation
Scenario testing

Chapter 5

Linear Programming

Learning Outcomes:

- Be able to formulate linear programming problems;
- Be able to solve linear programming problems using a graphical method;
- Be able to solve linear programming problems using the Simplex tableau method.

5.1 Introduction

Problems involving the maximisation and minimisation of functions have attracted the attention of mathematicians for centuries. The simplest of such problems is that of optimising a function of one variable using elementary calculus, i.e., if

$$y = f(x)$$

then the maximum / minimum values of y are given at the x which satisfies

$$f'(x) = 0.$$

The optimisation of a function of several variables can be found by setting the partial derivatives to zero. You may have seen this in your second year calculus courses.

An extension occurs when some of the variables are constrained, and the Lagrange multiplier technique may be used here. I.e.

$$\begin{aligned} &\text{maximise / minimise } f(x, y, u, v) \\ &\text{subject to} \\ &g(x, y, u, v) = 0 \\ &\text{and } h(x, y, u, v) = 0. \end{aligned}$$

During the Second World War a new optimisation problem emerged which could not be solved using any classical method. The problem was that of maximising or minimising some function subject to constraint conditions, but with the added complication that some of the variables had to be non-negative.

Therefore a lot of effort was placed on this problem mainly due to its usefulness in the solution of practical problems. In particular the case where the objective function and constraint equations are linear has received much attention. Such problems are known as *linear programming problems*. Here the word 'programming' refers to a structured algorithm, and not to a computer program.

A typical linear programming (LP) problem can be written as:

Maximise (or minimise)

$$f = \sum_{j=1}^{n} c_j x_j$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for all } i = 1, 2, \ldots, n$$

and the non-negativity condition

$$x_j \geq 0 \quad \text{for all } j = 1, 2, \ldots, n$$

Here the a_{ij} are called the left hand matrix values, b_i are called the right hand side constants, and c_j are called the cost function constants.

Note that LP problems are linear, and so do not include any non-linear terms such as x^3, $x_1 x_2$, $\sqrt{x}$, $\log(x_2)$, and so on.

We shall consider two methods of solving this problem, graphically, and using a Simplex tableau. Before this, we consider the task of problem formulation.

5.2 Formulation

In Operational Research we are interested in solving real-life problems. Linear programming is a technique we may apply to such problems in order to find optimal solutions. However we must first learn to formulate our practical problem into a mathematical problem. Consider the following example:

Example 18 *Consider:*

(a) You are in an eating competition that will last for one hour. You can either eat sausages that each take 2 minutes to eat, or burgers that each require 3 minutes to eat. You will receive 4 points for each sausage and 5 for each burger. How many of each type should you eat so as to maximise the number of points you score?

(b) What if there is an additional requirement that the number of you eat of one type cannot exceed the number you eat of the other type by more than 5.

Steps to formulating the problem:

1. Define the variables (clearly)
2. State the objective function (must be linear)
3. State the constraints (must be linear)
4. Include the non-negativity constraint
5. Decide whether variables should be integer, binary, etc.[1]

Solution to Example 18 *Let S be the number of sausages to eat, and let B be the number of burgers to eat. Then the linear programming problem becomes:*

Maximise:

$$4S + 5B \tag{5.1}$$

subject to

$$2S + 3B \leq 60 \tag{5.2}$$

$$S \leq B + 5 \tag{5.3}$$

$$B \leq S + 5 \tag{5.4}$$

$$S, B \geq 0 \tag{5.5}$$

Here (5.1) is the objective function that is to be maximised, representing the number of points gained; (5.2) is the time constraint representing the time needed to eat the items; (5.3) and (5.5) are constraints ensuring that no number of item exceeds five of the other; and (5.5) is the non-negativity constraint as we cannot eat a negative number of sausages or burgers.

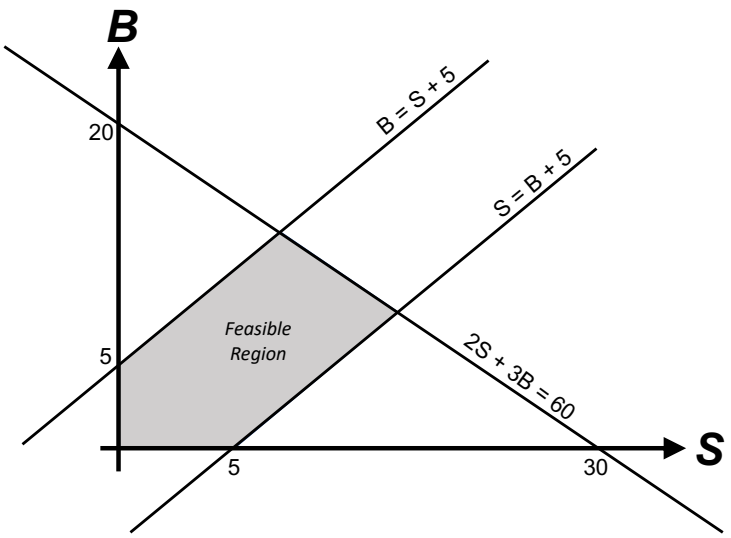
Note - it may be assumed that the number of burgers and sausages should be integer. For the purpose of this course we will not consider integer constraints - these will be considered in the Optimisation module next year.

[1] Note that integer and binary variables are not considered on this course but will be considered in the year 3 module on Optimisation.

5.3 Graphical Solution Method

This is a method of finding the values of the decision variables by drawing graphs, although we can only deal with problems with two variables in this way. The method is to draw the linear constraints in the Cartesian plane and identify the *feasible region*. This is the area where all the constraints are satisfied. Then we calculate the value of the objective function at each vertex that defines the feasible region, identifying the vertex that optimises the objective function.

Consider the problem in Example 18. Draw the graph by drawing the lines corresponding to the constraints:



The feasible region is the set of points (solutions) that satisfy all the constraints.

The optimal solution is given by the point that lies within the feasible region that optimises the objective function. It will always be on a vertex point that defines the feasible region. Here those points are given by: $(0, 0)$, $(0, 5)$, $(5, 0)$, the point where $B = S + 5$ meets $2S + 3B = 60$ i.e. $(9, 14)$, and the point where $S = B + 5$ meets $2S + 3B = 60$ i.e. $(15, 10)$.

If we calculate the value of the objective function at each point:

Point	Objective $= 4S + 5B$
$(0, 0)$	$4(0) + 5(0) = 0$
$(0, 5)$	$4(0) + 5(5) = 25$
$(5, 0)$	$4(5) + 5(0) = 20$
$(9, 14)$	$4(9) + 5(14) = 106$
$(15, 10)$	$4(15) + 5(10) = 110$

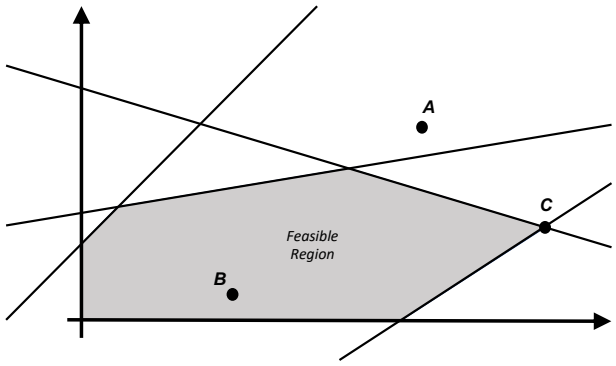
Therefore the optimal solution is gaining 110 points when eating 15 sausages and 10 burgers. Note that when presenting the optimal solution is should be given according to the context of the original question.

Investigating the Solution Space

The solution space is all the points $(x_1, x_2, \ldots, x_n)$. In two dimensions we have began to depict this space graphically as a Cartesian plane. Points within the solution space can be classified as either *basic* or *non-basic*, and as either *feasible* or *non-feasible*:

- **Feasible** points lie within the feasible region, that is they satisfy all the constraints. Otherwise the point is **non-feasible**.
- A feasible point is **basic** if it is a vertex of the polyhedron defining the feasible region, that is if n of the constraints are tight. Otherwise the point is **non-basic**.

Consider the graphic interpretation of a linear programming problem below:



The three points are classified as follows:

	Feasible	**Non-feasible**
Basic	C	
Non-Basic	B	A

Drawing a picture can also be useful in determining if the feasible region is **bounded** or **unbounded**. The previous example was bounded. Now consider:

Maximise:

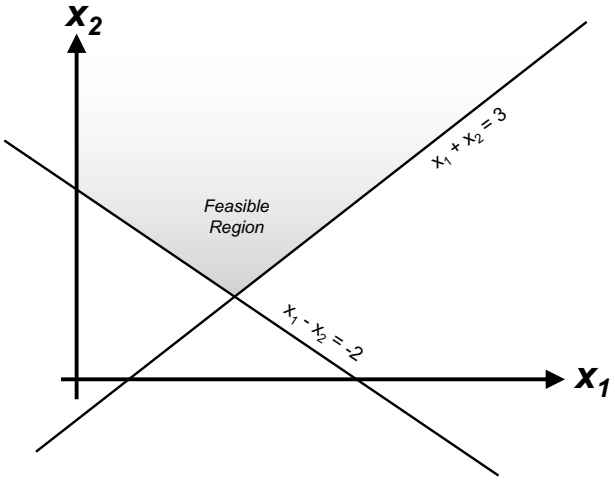
$$3x_1 + 5x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &\geq 3 \\ -x_1 + x_2 &\geq -2 \\ x_1, x_2 &\geq 0 \end{aligned}$$

The constraints now yield an unbounded region, and so the problem is unbounded. In this case as it is a maximisation problem, we can choose x_1 and x_2 to be as large as possible, as long as x_1 is 2 greater than x_2, and the constraints will still be satisfied, and so we can increase the value of the objective function infinitely.

In this course we will only be dealing with bounded problems.



Drawing a picture can also be useful in determining if the problem is feasible. Consider:

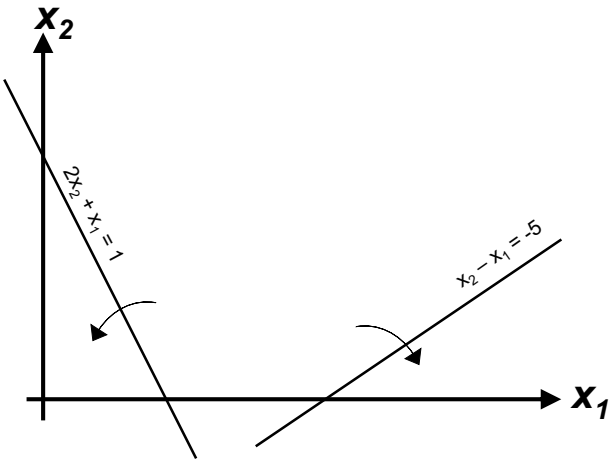
Maximise:

$$3x_1 + 5x_2$$

subject to

$$\begin{aligned} x_2 - x_1 &\leq -5 \\ x_2 + \frac{1}{2}x_1 &\geq 1 \\ x_1, x_2 &\geq 0 \end{aligned}$$

The image below shows that there is no feasible region (the arrows indicate which side of the line the satisfies constraints). Therefore the problem is **infeasible**.



Example 19 *The management of a coal electricity power plant need to comply with the latest emission standards under air pollution control laws. For the plant in question the maximum emission rates are 12 kg/hour. Two types of coal are available, Grade A which is hard, clean burning and domestically produced coal, and Grade B which is a cheap, soft, and smoky imported coal.*
First coal is brought to the plant by railway and is carried to the pulveriser unit via a conveyor system. The conveyor loading system has a maximum capacity of 20 tons of coal per hour.
Then the coal is pulverised and fed directly into the combustion chamber. Due to the hardness of the domestic coal the maximum capacity of this coal on the pulveriser unit is 16 tons per hour, whereas it is 24 tons per hour for the imported coal.
The particulate emissions per type of coal are:

- *Grade A: 0.5 kg emission / ton burned*
- *Grade B: 1.0 kg emission / ton burned*

The steam produced is by Grade A coal produces £24,000 per ton, whereas steam produced by Grade B produces £20,000 pounds per ton.
The government places a restriction on the amount of imported coal that can be used. The restriction is that for every ton of imported coal used, the company must use at least 1.5 tons of domestic coal.
Management want to know: "Given the situation described above, what is the maximum possible output of electricity of the plant, and how would they achieve that?"

Solution to Example 19 *Formulation: Let x_1 be the number of tons of Grade A coal used per hour, and let x_2 be the number of tons of Grade B coal used per hour. Then the linear programming problem becomes:*

Maximise:

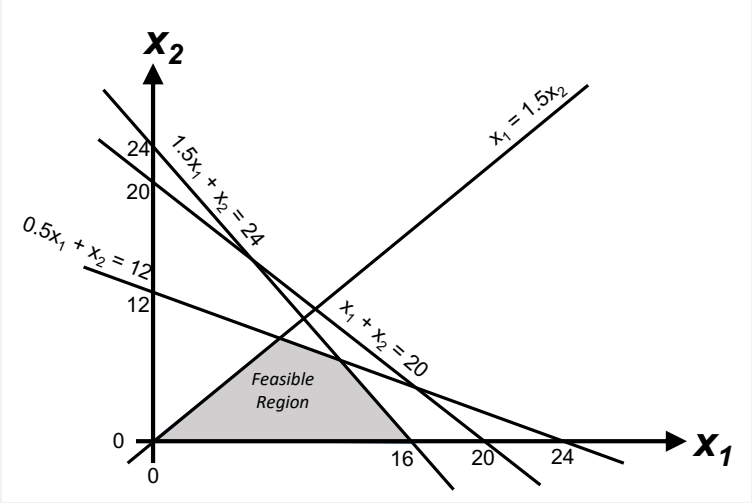
$$24000x_1 + 20000x_2 \tag{5.6}$$

subject to

$$x_1 + x_2 \leq 20 \tag{5.7}$$
$$0.5x_1 + x_2 \leq 12 \tag{5.8}$$
$$\frac{1}{16}x_1 + \frac{1}{24}x_2 \leq 1 \tag{5.9}$$
$$x_1 - 1.5x_2 \geq 0 \tag{5.10}$$
$$x_1, x_2 \geq 0 \tag{5.11}$$

Here (5.6) is the objective function representing the total profit produced per hour. The constraints: (5.7) represents the conveyor loading capacity; (5.8) ensures that hourly emissions are not exceeded; (5.9) is the hourly capacity of the pulveriser; (5.10) enforces using at least 1.5 times more domestic coal than imported coal; and (5.11) indicates that we cannot use a negative amount of each coal type.

Solution to Example 19 (continuing from p. 80) *Drawing this gives:*



There are four vertices of the feasible region: $(0,0)$, $(16,0)$, *the point where* $\frac{3}{2}x_1 + x_2 = 24$ *meets* $0.5x_1 + x_2 = 12$ *i.e.* $(12,6)$, *and the point where* $0.5x_1 + x_2 = 12$ *meets* $x_1 = 1.5x_2$ *i.e.* $\left(\frac{72}{7}, \frac{48}{7}\right)$.
If we calculate the value of the objective function at each point:

Point	*Objective* $= 24000x_1 + 20000x_2$
$(0,0)$	$24000(0) + 20000(0) = 0$
$(16,0)$	$24000(16) + 20000(0) = 384000$
$(12,6)$	$24000(12) + 20000(6) = 408000$
$\left(\frac{72}{7}, \frac{48}{7}\right)$	$24000\left(\frac{72}{7}\right) + 20000\left(\frac{48}{7}\right) = 384000$

Therefore the optimal solution is producing 384000 *pounds of steam when using* 12 *tons of Grade A coal per hour and* 6 *tons of Grade B coal per hour.*

Consider another example:

Example 20 *A builder wishes to develop a ten acre site by constructing two types of houses:*

- *Type A - costs £56,000, on which 10% profit can be made. It uses* $\frac{1}{10}$*th of an acre.*
- *Type B - costs £80,000, on which 20% profit can be made. It uses a* $\frac{1}{4}$ *of an acre.*

The council require that a maximum of 8 houses per acre be built.

(a) How many of each house should be built in order to maximise profit?
(b) If the profit from the cheaper house changes from 10% to 15% what is the new solution?

Solution to Example 20 *Formulation: Let x_1 be the number of Type A houses built per acre, and let x_2 be the number of Type B houses built per acre. Then the original linear programming problem becomes:*

Maximise:

$$5600x_1 + 16000x_2 \tag{5.12}$$

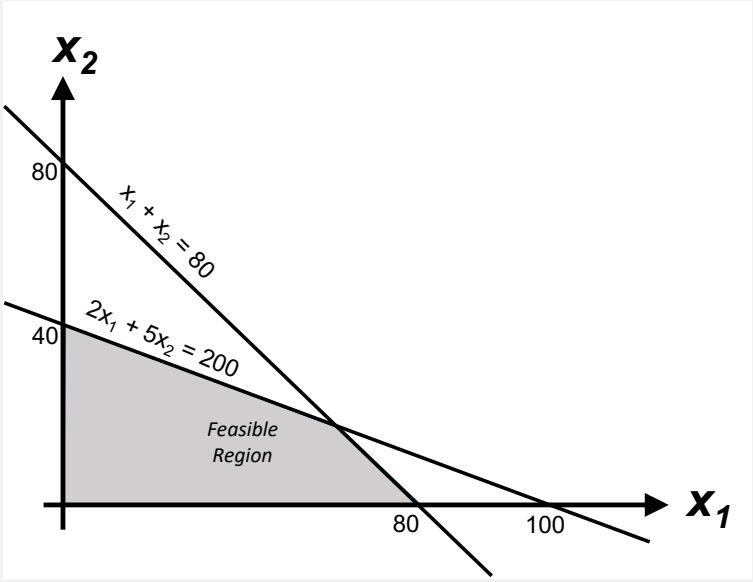
subject to

$$x_1 + x_2 \leq 80 \tag{5.13}$$

$$\frac{1}{10}x_1 + \frac{1}{4}x_2 \leq 10 \tag{5.14}$$

$$x_1, x_2 \geq 0 \quad \textit{and integer.} \tag{5.15}$$

Here: (5.12) is the objective function that is to be maximised, representing the total profit. The constraints are: (5.13) representing the council's houses per acre limit; (5.14) makes sure that there is sufficient space to build the houses; and (5.15) is the non-negativity and integer constraint, indicating that we cannot build negative or non-integer numbers of houses.
For part (b), the objective function becomes $8400x_1 + 16000x_2$.
Now drawing this gives:



Solution to Example 20 (continuing from p. 82) *There are four vertices of the feasible region:* $(0,0)$, $(0,80)$, $(40,0)$, *and the point where* $x_1 + x_2 = 80$ *meets* $2x_1 + 5x_2 = 200$ *i.e.* $\left(\frac{200}{3}, \frac{40}{3}\right)$.
If we calculate the value of the objective function at each point, for both objective function scenarios:

Point	*Objective (a)* $5600x_1 + 16000_2$	*Objective (b)* $8400x_1 + 16000x_2$
$(0,0)$	0	0
$(80,0)$	448000	672000
$(0,40)$	640000	640000
$\left(\frac{200}{3}, \frac{40}{3}\right)$	586666.67	773333.33

In scenario (a) the optimal solution gives a profit of £640,000 when building 0 *houses of Type A and* 40 *houses of Type B.*
In scenario (b) the optimal solution gives a profit of £773,333.33 when building $\frac{200}{3}$ *houses of Type A and* $\frac{40}{3}$ *houses of Type B.*

Note that we cannot build $\frac{200}{3}$ houses of Type A as we cannot build a non-integer number of houses. This was an example of an *integer linear programming problem*, and there are techniques available to solve these, such as cutting planes and branch-and-bound algorithms, but they are not covered in this module. However, solving this and relaxing the integer constraint can still be useful. This is called the *LP relaxation* problem, and gives us an upper-bound for the true optimal profit. That is, we know in scenario (b) that we cannot obtain a profit greater than £773,333.33 if building an integer number of houses.

5.4 Standard Form

In general we can write linear programming problems in a standard way, called the standard form of a linear programming problem. To write them in standard form, we need:

- All variables are restricted to be non-negative,
- All constraints are *equalities*,
- All constraints have a constant, non-negative, right hand side.

So far we have seen that constraints are *inequalities*, but we can re-write them as equalities, by introducing additional *slack variables* $s_i \geq 0$. These variables measure the slack in each constraint (how far away from equality are they).

Consider the following linear programming problem that is not written in standard form:

Maximise:

$$10A + 5B - C$$

subject to

$$\begin{aligned} A + B &\leq 6 \\ A - C &\geq -2 \\ A + B - C &\geq 3 \\ B &\leq C \\ A, B, C &\geq 0 \end{aligned}$$

In standard form, we need to introduce four slack variables, s_1, s_2, s_3, and s_4, one for each constraint, to convert the inequalities to equalities. Knowing that these variables are restricted to be positive, we must either subtract or add the slack variables. We must also rearrange some equations to achieve a constant, non-negative, right hand side:

Maximise:

$$10A + 5B - C$$

subject to

$$\begin{aligned} A + B + s_1 &= 6 \\ -A + C + s_2 &= 2 \\ A + B - C - s_3 &= 3 \\ B - C + s_4 &= 0 \\ A, B, C &\geq 0 \end{aligned}$$

Matrix Notation

It is sometimes useful to write a linear programmming problem using matrix notation. Letting $\mathbf{x}$ be a vector of decision variables, $\mathbf{c}$ a vector of the coefficients in the objective function, $\mathbf{A}$ be a matrix of coefficients of the constraints (rows corresponding to each constraint, columns corresponding to each decision variable), and $\mathbf{b}$ correspond to the values of the right hand sides of the contraint inequalities, then:

Maximise:

$$\mathbf{c}^T\mathbf{x}$$

subject to

$$\begin{aligned} \mathbf{Ax} &\leq \mathbf{b} \\ \mathbf{x} &\geq 0 \end{aligned}$$

For the example linear programming problem in Example 19:

Maximise:

$$24000x_1 + 20000x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 20 \\ \frac{1}{2}x_1 + x_2 &\leq 12 \\ \frac{1}{16}x_1 + \frac{1}{24}x_2 &\leq 1 \\ -x_1 + \frac{3}{2}x_2 &\leq 0 \\ x_1, x_2 &\geq 0 \end{aligned}$$

in matrix form, the vectors become:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 24000 \\ 20000 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 20 \\ 12 \\ 1 \\ 0 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1/2 & 1 \\ 1/16 & 1/24 \\ -1 & 3/2 \end{bmatrix}$$

5.5 Simplex Tableau

Graphical solution methods can only be used when there are two variables (it is very hard to draw any more than two dimensions!). The Simplex tableau method, sometimes called the Simplex algorithm, is a method that can solve linear programming problems with many variables. Consider the linear programming problem we formulated in Example 18:

Maximise:

$$4S + 5B$$

subject to

$$\begin{aligned} 2S + 3B &\leq 60 \\ S &\leq B + 5 \\ B &\leq S + 5 \\ S, B &\geq 0 \end{aligned}$$

We can re-write the inequalities as equalities, by introducing additional slack variables $s_1, s_2, s_3 \geq 0$. Therefore the constraints are written as:

$$2S + 3B + s_1 = 60 \tag{5.16}$$
$$S - B + s_2 = 5 \tag{5.17}$$
$$-S + B + s_3 = +5 \tag{5.18}$$

Now the whole linear programming problem can be re-written in a tableau as follows:

	S	B	s_1	s_2	s_3
60	2	3	1	0	0
5	1	-1	0	1	0
5	-1	1	0	0	1
0	-4	-5	0	0	0

Note that when dealing with a maximisation problem, the objective function is written in the tableau as the negative of the true objective function, hence there the coefficient of S is written as -4 rather than 4, and the coefficient of B is written as -5 rather than 5.

- **Basic variables** are those which correspond to a column consisting of a single '1' and the rest '0's. They have values ≥ 0 equal to the value in the first column in the row corresponding to its '1'.
- **Non-basic variables** are the remaining variables, and are equal to 0.

In the above tableau, S and B are non-basic variables, and so $S = B = 0$. The basic variables are s_1, s_2 and s_3, with values $s_1 = 60$, $s_2 = 5$ and $s_3 = 5$. Notice these values satisfy Equations 5.16, 5.17 and 5.18; and correspond to an objective function value of 0, seen in the final row of the first column.

The aim now of the Simplex algorithm to to undertake specific row operations to find the basic variables and their values that maximise the objective function value. This is equivalent to traversing the vertices of the feasible region.

Simplex Algorithm

Steps:

1. First we look in the objective row (the final row) for the column with the **most negative value**. If there is no negative value, then the solution is optimal. The chosen column is called the *work column*.
2. Next examine the **positive** coefficients in this column and select the row with the **least positive ratio** when taking the constants (first) column and dividing by it. The chosen element is called the *pivot*, and we circle or star it.
3. Divide the row of the pivot by the pivot.
4. By choosing multipliers of the new row, subtract it from the remaining rows making every other element 0 in the work column.
5. Repeat the above until there are no negative values in the objective row.

For the tableau given by the problem in Example 18 the 3rd column becomes the work column and the third row becomes the pivot row: circle the pivot and divide the row by the pivot:

	S	B	s_1	s_2	s_3
60	2	3	1	0	0
5	1	-1	0	1	0
5	-1	(1)	0	0	1
0	-4	-5	0	0	0

Now perform row operations to get make the work column a basic variable. Here $r_1 = r_1 - 3r_3$, $r_2 = r_2 + r_3$, and $r_4 = r_4 + 5r_3$. We then identify the next pivot:

	S	B	s_1	s_2	s_3
45	(5)	0	1	0	-3
10	0	0	0	1	1
5	-1	1	0	0	1
25	-9	0	0	0	5

Now divide the pivot row by the pivot:

	S	B	s_1	s_2	s_3
9	(1)	0	$1/5$	0	$-3/5$
10	0	0	0	1	1
5	-1	1	0	0	1
25	-9	0	0	0	5

Perform $r_3 = r_3 + r_1$ and $r_4 = r_4 + 9r_1$, and identify the next pivot:

	S	B	s_1	s_2	s_3
9	1	0	$1/5$	0	$-3/5$
10	0	0	0	1	(1)
14	0	1	$1/5$	0	$2/5$
106	0	0	$9/5$	0	$-2/5$

Perform $r_1 = r_1 + \frac{3}{5}r_2$, $r_3 = r_3 - \frac{2}{5}r_2$, and $r_4 = r_4 + \frac{2}{5}r_2$:

	S	B	s_1	s_2	s_3
15	1	0	$1/5$	$3/5$	0
10	0	0	0	1	1
10	0	1	$1/5$	$-2/5$	0
110	0	0	$9/5$	$2/5$	0

Now there are no negative values in the objective row, and so we can read off the solution. The basic variables are S, B and s_3, with values $S = 15$, $B = 10$ and $s_3 = 10$. The non-basic variables are s_1 and s_2, indicating that constraints 5.2 and 5.3 have become equalities. The objective function value is 110, indicated by the lower left most cell, and confirmed by placing substituting the values for the basic variables into the objective function: $(4\times15)+(5\times10) = 110$. Therefore the solution is to eat 15 sausages and 10 burgers.

Note:

- If any slack variables remain basic, then their corresponding constraints are considered slack, and the value of the variable indicates the amount of slack in that particular constraint.
- If any slack variables are zero, then the corresponding constraint is tight.
- The coefficients of any non-basic variables in the objective row are called the *shadow price* of their corresponding constraints. It is the maximum amount that would be worth paying to increase the right hand side of that constraint by one unit.

Let's solve Example 20 above, this time using the Simplex method:

Example 21 *Solve the following linear programming problem using the Simplex tableau method:*

Maximise:

$$5600x_1 + 16000x_2$$

subject to

$$x_1 + x_2 \leq 80$$
$$2x_1 + 5x_2 \leq 200$$
$$x_1, x_2 \geq 0$$

Solution to Example 21 *The initial tableau and pivot is given by:*

	x_1	x_2	s_1	s_2
80	*1*	*1*	*1*	*0*
200	*2*	*(5)*	*0*	*1*
0	*-5600*	*-16000*	*0*	*0*

Now perform $r_1 = r_1 - \frac{1}{5}r_2$, $r_2 = \frac{1}{5}r_2$, *and* $r_3 = 3200r_2$*:*

	x_1	x_2	s_1	s_2
40	$3/5$	*0*	*1*	$1/5$
40	$2/5$	*1*	*0*	$1/5$
640000	*800*	*0*	*0*	*3200*

And we are done. We read off the solution as: $x_1 = 0$, $x_2 = 40$, *and the maximum profit as £640,000.*

Tips:

- *There should never be and negative values in the left hand side column:* The values of the left hand column represent the current values of the basic variables, including any slack variables. The simplex algorithm cycles through the vertices of the feasible region, and so no basic or slack variable would be negative in this region. If you encounter a negative value in the left hand column, you have made a mistake!
- *Ties:* If there are any ties when choosing a pivot, make an arbitrary choice! This is equivalent to going in one of two directions when cycling the vertices of the feasible region, either way we will reach the destination. Maybe one direction will take longer (more iterations) to reach an optimal solution, but we have no way of knowing that yet.
- *Zeros in the left hand column:* If we have a situation with zeros in the left hand column, then we have basic variables that are set to zero. This situation is called *degeneracy*.

5.6 Minimisation Problems

The Simplex algorithm can be used for minimisation problems too. Consider that:

$$\text{Minimise} \quad f(\underline{\mathbf{x}})$$

is equivalent to

$$\text{Maximise} \quad -f(\underline{\mathbf{x}})$$

and so we only need to write the Simplex tableau with the objective row coefficients written as their positive values.

Example 22 *Solve the following minimisation linear programming problem using the Simplex tableau method:*

Minimise:

$$4x_1 - 3x_2$$

subject to

$$\begin{aligned} x_1 - x_2 &\leq 5 \\ 2x_1 - x_2 &\leq 10 \\ x_1 + x_2 &\leq 10 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Solution to Example 22 *The initial tableau and pivot is given by:*

	x_1	x_2	s_1	s_2	s_3
5	1	-1	1	0	0
10	2	-1	0	1	0
10	1	(1)	0	0	1
0	4	-3	0	0	0

Notice the coefficients in the objective row are now written as 4 *and* -3 *as it is a minimisation problem, rather than* -4 *and* 3.

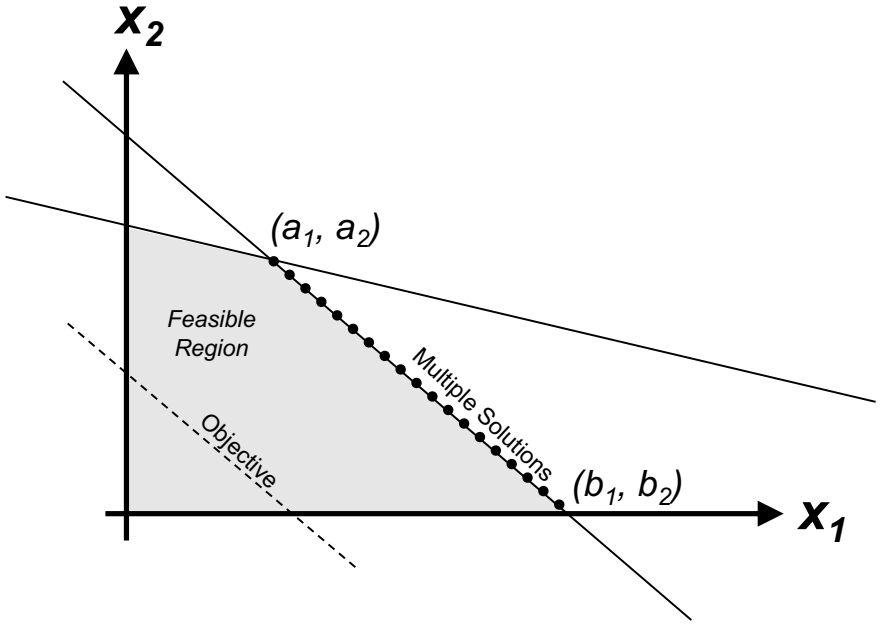
Now perform $r_1 = r_1 + r_3$, $r_2 = r_2 + r_3$, *and* $r_4 = r_4 + 3r_3$ *to get:*

	x_1	x_2	s_1	s_2	s_3
15	2	0	1	0	1
20	3	0	0	1	1
10	1	1	0	0	1
30	7	0	0	0	3

And we are done. We read off the solution as: $x_1 = 0$, $x_2 = 10$, *and the minimum value of the objective function is* 30.

5.7 Non-Unique Solutions

If the objective function and one of the constraints are parallel to one another, then there is the possibility of having multiple optimal solutions. In these cases, all points along the edge of the feasible region parallel to the objective function would be optimal solutions, each yielding the same value of the objective function.



If two vertex solutions are found, $\underline{\mathbf{a}} = (a_1, a_2, \ldots, a_n)$ and $\underline{\mathbf{b}} = (b_1, b_2, \ldots, b_n)$, then the set of all optimal solutions is given by $\{(1-t)\underline{\mathbf{a}} + t\underline{\mathbf{b}} \text{ for all } t \in [0,1]\}$.

When carrying out the Simplex algorithm, multiple solutions are indicated by non-basic variables having zeros as coefficients in the objective row when no other coefficient is negative. Once this is reached, find the next optimal vertex by using that non-basic variable as a work column and pivot one more time.

As an example:

Example 23 *Find all possible optimal solutions to the following linear programming problem:*

Maximise:

$$3x_1 + 3x_2$$

subject to

$$\begin{aligned} x_1 - x_2 &\leq 5 \\ 2x_1 - x_2 &\leq 10 \\ x_1 + x_2 &\leq 10 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Solution to Example 23 *The initial tableau and pivot is given by:*

	x_1	x_2	s_1	s_2	s_3
5	1	-1	1	0	0
10	2	-1	0	1	0
10	1	(1)	0	0	1
0	-3	-3	0	0	0

Now perform $r_1 = r_1 + r_3$, $r_2 = r_2 + r_3$, *and* $r_4 = r_4 + 3r_3$ *to get:*

	x_1	x_2	s_1	s_2	s_3
15	2	0	1	0	1
20	(3)	0	0	1	1
10	1	1	0	0	1
30	0	0	0	0	3

There are no negative coefficients in the bottom row, so we have found at least one optimal solution. We read off the solution as: $x_1 = 0$, $x_2 = 10$, *and the maximum value of the objective function is* 30.

However, x_1 *is a non-basic variable, and it's coefficient in the objective row is zero. Therefore we know there is another optimal solution. Let's use this column as the work column, using the circled element as the next pivot. We perform* $r_1 = r_1 - \frac{2}{3}r_2$, $r_2 = \frac{1}{3}r_2$, $r_3 = r_3 - \frac{1}{3}r_2$*:*

	x_1	x_2	s_1	s_2	s_3
$5/3$	0	0	1	$-2/3$	$1/3$
$20/3$	1	0	0	$1/3$	$1/3$
$10/3$	0	1	0	$-1/3$	$2/3$
30	0	0	0	0	3

And we have found another optimal solution: $x_1 = 20/3$, $x_2 = 10/3$, *and the maximum value of the objective function of course remains at* 30.

Therefore all optimal solutions are given by $\left\{(x_1, x_2) = \left(\frac{20}{3}t, 10 - \frac{20}{3}t\right) \quad \text{for all} \quad t \in [0,1]\right\}$.

5.8 Greater Than Constraints ($\geq$)

So far we have only considered linear programming problems where all constraints (except the non-negativity constraints) are written as less than or equal to constraints ($\leq$). This allowed us to convert the inequality constraints into equalities by adding positive slack variables. Now consider a greater than or equal to constraint, for example

$$x_1 + x_2 \geq 80$$
$$x_1 - 2x_2 \geq 10$$

now introducing the slack variable we would need to subtract it, rather than add it: $x_1+x_2-s_1 = 80$. However this would now imply that we begin with a negative basic variable $s_1 = -80$, which we cannot do as it violates our non-negativity condition. That is, our initial starting vertex $x_1 = x_2 = 0$ is not feasible as $0 + 0 \ngeq 80$.

We need to find a feasible initial solution before we begin to carry out the Simplex method. But we can use the Simplex method itself to do this. We introduce *artificial variables* like so:

$$x_1 + x_2 - s_1 + a_1 = 80$$
$$x_1 - 2x_2 - s_2 + a_2 = 10$$

and we find an initial feasible solution by minimising the sum of artificial variables $a_1 + a_2$, which we can do using the Simplex method. This is called the **two-phase approach**:

Two-Phase Approach

1. Phase 1:
   - Write constraints with both slack and artificial variables.
   - Write the new objective function (the sum of the artificial variables) in terms of the non-basic variables.
   - Write the Simplex tableau with both objective functions.
   - Solve the Simplex tableau in terms of the new objective function.
2. Phase 2:
   - Remove the temporary objective row and the columns corresponding to the artificial variables.
   - Solve the Simplex tableau in terms of the original objective function.

Let's do an example:

Example 24 *Solve:*

Minimise:

$$2x_1 + 5x_2$$

subject to

$$3x_1 + 4x_2 \geq 12$$
$$5x_1 + 2x_2 \geq 8$$
$$x_1, x_2 \geq 0$$

Solution to Example 24 *Phase 1: We first rewrite the constraints as equalities:*

$$3x_1 + 4x_2 - s_1 + a_1 = 12$$
$$5x_1 + 2x_2 - s_2 + a_2 = 8$$

Then, combining and rearranging these can write the new objective function as:

$$a_1 + a_2 = 20 - 8x_1 - 6x_2 + s_1 + s_2$$

And so the new objective function becomes minimising $-8x_1 - 6x_2 + s_1 + s_2$, whose current value is 20. Write down the initial tableau with both objectives (but only solve in terms of this new objective):

	x_1	x_2	s_1	s_2	a_1	a_2
12	3	4	-1	0	1	0
8	(5)	2	0	-1	0	1
-20	-8	-6	1	1	0	0
0	2	5	0	0	0	0

Apply $r_1 = r_1 - \frac{3}{5}r_2$, $r_2 = \frac{1}{5}r_2$, $r_3 = r_3 + \frac{8}{5}r_2$, and $r_4 = r_4 - \frac{2}{5}r_2$:

	x_1	x_2	s_1	s_2	a_1	a_2
$36/5$	0	$(14/5)$	-1	$3/5$	1	$-3/5$
$8/5$	1	$\frac{2}{5}$	0	$-1/5$	0	$1/5$
$-36/5$	0	$-14/5$	1	$-3/5$	0	$8/5$
$-16/5$	0	$21/5$	0	$2/5$	0	$-2/5$

Apply $r_1 = \frac{5}{14}r_1$, $r_2 = r_2 - \frac{1}{7}r_1$, $r_3 = r_3 + r_1$, and $r_4 = r_4 - \frac{3}{2}r_1$:

	x_1	x_2	s_1	s_2	a_1	a_2
$18/7$	0	1	$-5/14$	$3/14$	$5/14$	$-3/14$
$4/7$	1	0	$1/7$	$-2/7$	$-1/7$	$2/7$
0	0	0	0	0	1	1
-14	0	0	$3/2$	$-1/2$	$-3/2$	$-1/2$

Now the first objective row has no negative values, and so we have finished Phase 1.

Solution to Example 24 (continuing from p. 94) *Phase 2: Remove the appropriate columns and rows:*

	x_1	x_2	s_1	s_2	a_1	a_2
$18/7$	0	1	$-5/14$	$3/14$	$5/14$	$-3/14$
$4/7$	1	0	$1/7$	$-2/7$	$-1/7$	$2/7$
~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~1~~
-14	0	0	$3/2$	$-1/2$	$-3/2$	$-1/2$

which gives:

	x_1	x_2	s_1	s_2
$18/7$	0	1	$-5/14$	$3/14$
$4/7$	1	0	$1/7$	$-2/7$
-14	0	0	$3/2$	$-1/2$

Apply $r_1 = \frac{14}{3}r_1$, $r_2 = r_2 + \frac{4}{3}r_1$, *and* $r_3 = r_3 + \frac{7}{3}r_1$*:*

	x_1	x_2	s_1	s_2
12	0	$14/3$	$-5/3$	1
4	1	$4/3$	$-1/3$	0
-8	0	$7/3$	$2/3$	0

And we are done, we can read off the optimal solution as $x_1 = 4$, $x_2 = 0$, *with the objective function having a value of* 8.

5.9 Equality Constraints

Similar to greater-than constraints, artificial variables are also required when dealing with equality constraints. Consider the constraint:

$$2x_1 + 5x_2 = 9$$

This is equivalent to the following *pair* of constraints:

$$2x_1 + 5x_2 \leq 9 \tag{5.19}$$
$$2x_1 + 5x_2 \geq 9 \tag{5.20}$$

Which, adding the appropriate slack and artificial variables as before, we get:

$$\begin{aligned} 2x_1 + 5x_2 + s_1 &= 9 \\ 2x_1 + 5x_2 - s_2 + a_1 &= 9 \end{aligned}$$

However, note that in this case slack variables do not make sense. Both Constraints 5.19 and 5.19 must be tight constraints in order for the equality constraint to hold. Therefore $s_1 = s_2 = 0$ by definition, and as such do not really exist for our problem. So for equality constraints, we do not require slack variables, but only need the artificial variable in order for Simplex method to be able to begin with enough basic variables. That is:

$$2x_1 + 5x_2 + a_1 = 9$$

Let's do an example:

Example 25 *Solve:*

Maximise:

$$2x_1 + 5x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &= 4 \\ 2x_1 + x_2 &\geq 5 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Solution to Example 25 *Phase 1: We first rewrite the constraints as equalities:*

$$\begin{aligned} x_1 + x_2 + a_1 &= 4 \\ 2x_1 + x_2 - s_1 + a_2 &= 5 \end{aligned}$$

Then, combining and rearranging these can write the new objective function as:

$$a_1 + a_2 = 9 - 3x_1 - 2x_2 + s_1$$

And so the new objective function becomes minimising $-3x_1 - 2x_2 + s_1$, whose current value is 9. Write down the initial tableau with both objectives (but only solve in terms of this new objective):

	x_1	x_2	s_1	a_1	a_2
4	*1*	*1*	*0*	*1*	*0*
5	*(2)*	*1*	*-1*	*0*	*1*
-9	*-3*	*-2*	*1*	*0*	*0*
0	*-2*	*-5*	*0*	*0*	*0*

Solution to Example 25 (continuing from p. 96) *Apply* $\bar{r}_2 = \frac{1}{2}r_2$, $r_1 = r_1 - \bar{r}_2$, $r_3 = r_3 + 3\bar{r}_2$, *and* $r_4 = r_4 + r_2$*:*

	x_1	x_2	s_1	a_1	a_2
$3/2$	0	$1/2$	$1/2$	1	$-1/2$
$5/2$	1	$1/2$	$-1/2$	0	$1/2$
$-3/2$	0	$-1/2$	$-1/2$	0	3
5	0	-4	-1	0	1

Apply $\bar{r}_1 = 2r_1$, $r_2 = r_2 - r_1$, $r_3 = r_3 + 4\bar{r}_1$, *and* $r_4 = r_4 + r_1$*:*

	x_1	x_2	s_1	a_1	a_2
3	0	1	1	1	-1
1	1	0	-1	-1	-1
0	0	0	0	1	$5/2$
17	0	0	3	3	-2

Now the first objective row has no negative values, and so we have finished Phase 1.
Phase 2: Remove the appropriate columns and rows:

	x_1	x_2	s_1	a_1	a_2
3	0	1	1	1	-1
1	1	0	-1	-1	-1
~~0~~	~~0~~	~~0~~	~~0~~	1	~~$5/2$~~
17	0	0	3	3	-2

which gives:

	x_1	x_2	s_1
3	0	1	1
1	1	0	-1
17	0	0	3

There are no negative coefficients in the objective row, so we are done. We can read off the optimal solution as $x_1 = 1$, $x_2 = 3$, *with the objective function having a value of* 17.

Equality constraints may also be thought of in another way: as dimension reduction. For example if we have N decision variables, and one equality constraint, that means we can express one of the decision variables as some linear combination of the others. And so really we only have $N - 1$ decision variables. E.g. consider:

Maximise:

$$X_1 + X_2 + 8X_3$$

subject to

$$\begin{aligned} X_1 + 4X_2 &\leq 4 \\ 3X_1 - X_3 &\leq 5 \\ X_1 - X_2 - X_3 &= 0 \\ X_1, X_2, X_3 &\geq 0 \end{aligned}$$

The final constraint implies that $X_1 = X_2 + X_3$, and so we can use this substitution to reduce the number of decision variables to:

Maximise:

$$2X_2 + 9X_3$$

subject to

$$\begin{aligned} 5X_2 + X_3 &\leq 4 \\ 3X_2 + 2X_3 &\leq 5 \\ X_2, X_3 &\geq 0 \end{aligned}$$

5.10 Solving with Python

There exist specialised computer programmes that are designed specifically for solving linear programming problems. Examples include COIN-OR LP, CPLEX, Gurobi and XPress. Some of these are commercial software and require paid for licenses. `PuLP` is a Python library that interfaces with solvers such as these, is free to use, readable, and allows us to solve linear programming problems within a Python environment.

Consider the following example:

Example 26 *Use `PuLP` to solve the following linear programming problem:*

Maximise:

$$50A + 60B$$

subject to

$$\begin{aligned} 4A + 6B &\leq 24 \\ 5A + 4B &\leq 20 \\ A, B &\geq 0 \end{aligned}$$

Now adapt the code so that A and B must take on integer values.

Solution to Example 26 *The code would be:*

```
>>> import pulp
>>> # Create problem
>>> prob = pulp.LpProblem("Paint", pulp.LpMaximize)
>>> # Create decision variables
>>> paint = pulp.LpVariable.dicts("x", range(2))
>>> # Objective function
>>> objective_function = (5 * paint[0]) + (6 * paint[1])
>>> prob += objective_function
>>> # Constraints
>>> prob += (4 * paint[0]) + (6 * paint[1]) <= 24
>>> prob += (5 * paint[0]) + (4 * paint[1]) <= 20
>>> # Solve
>>> prob.solve()
>>> print(paint[0].value(), paint[1].value())
1.7142857, 2.8571429
```

If we required integer variables, we would specify that when creating the decision variables:

```
>>> prob = pulp.LpProblem("Paint", pulp.LpMaximize)
>>> paint = pulp.LpVariable.dicts("x", range(2), cat="Integer")
>>> objective_function = (5 * paint[0]) + (6 * paint[1])
>>> prob += objective_function
>>> prob += (4 * paint[0]) + (6 * paint[1]) <= 24
>>> prob += (5 * paint[0]) + (4 * paint[1]) <= 20
>>> prob.solve()
>>> print(paint[0].value(), paint[1].value())
0.0, 4.0
```

5.11 Further Formulation Examples

Example 27 *Pontypandy is considering a new bus system. Demand for the number of buses is different throughout the day, as shown in the following table:*

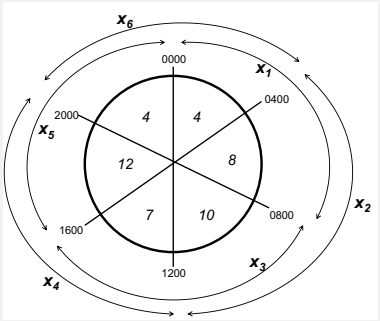
Time	*0000-0400*	*0400-0800*	*0800-1200*	*1200-1600*	*1600-2000*	*2000-0000*
Number of buses needed	*4*	*8*	*10*	*7*	*12*	*4*

Legislation says that each bus can only drive for 8 hours a day in contiguous shifts. What is the minimum number of buses that Pontypandy needs, and how should they be scheduled?

Solution to Example 27 *Let:*

- x_1 *be the number of buses on shift from 0000-0800,*
- x_2 *be the number of buses on shift from 0040-1200,*
- x_3 *be the number of buses on shift from 0080-1600,*
- x_4 *be the number of buses on shift from 1200-2000,*
- x_5 *be the number of buses on shift from 1600-0000, and*
- x_6 *be the number of buses on shift from 2000-0400.*

This is illustrated in the diagram below:



Then the linear programming problem becomes:

Minimise: $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$
subject to

$$\begin{array}{rcrcrcrcrcl}
x_1 & & & & & & & + & x_6 & \geq & 4 \\
x_1 & + & x_2 & & & & & & & \geq & 8 \\
 & & x_2 & + & x_3 & & & & & \geq & 10 \\
 & & & & x_3 & + & x_4 & & & \geq & 7 \\
 & & & & & & x_4 & + & x_5 & \geq & 12 \\
 & & & & & & & & x_5 & + & x_6 \quad \geq \quad 4
\end{array}$$

and $x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$ *and integer.*

Example 28 *A hospital ward has n nurses. The ward has morning shifts and afternoon shifts. It closes over night and is open 7 days per week. The ward manager wishes to produce a work roster such that:*

- *The number of nurses on each shift is at least equal to the minimum requirement,*
- *Full time nurses do 5 shifts per week, and part time nurses do 3 or 4 shifts per week,*
- *No nurse works both a morning and afternoon shift in the same day, and*
- *A nurse can request a particular day off.*

Formulate the linear programming problem. How many decision variables are there? How many constraints?

Solution to Example 28 *Let:*

- *m_j be the minimum nurse requirement for a morning shift on day j,*
- *a_j be the minimum nurse requirement for an afternoon shift on day j,*
- *f_j be a parameter indicating if nurse i is full time (1 if yes, 0 if no),*
- *x_{ij} be a binary decision variable indicating that nurse i works a morning shift on day j,*
- *y_{ij} be a binary decision variable indicating that nurse i works an afternoon shift on day j.*

Then the linear programming problem becomes:

Minimise:

$$\sum_i \sum_j (x_{ij} + y_{ij}) \tag{5.21}$$

subject to

$$\sum_i x_{ij} \leq m_j \text{ for all } j \tag{5.22}$$

$$\sum_i y_{ij} \leq a_j \text{ for all } j \tag{5.23}$$

$$y_{ij} + x_{ij} \leq 1 \text{ for all } i, j \tag{5.24}$$

$$\sum_j f_i (x_{ij} + y_{ij}) = 5 \text{ for all } i \tag{5.25}$$

$$\sum_j (1 - f_i)(x_{ij} + y_{ij}) \leq 4 \text{ for all } i \tag{5.26}$$

$$\sum_j (1 - f_i)(x_{ij} + y_{ij}) \geq 3 \text{ for all } i \tag{5.27}$$

$$x_{ij}, y_{ij} \text{ binary for all } i, j \tag{5.28}$$

Solution to Example 28 (continuing from p. 101) *Here (5.21) is the objective function that is to be minimised, ensuring no more nurses are scheduled than are needed. The constraints are: (5.22) ensures there are enough nurses on duty in the morning; (5.23) ensures there are enough nurses on duty in the afternoon; (5.25) ensures full-time nurses have 5 shifts; (5.26) and (5.27) ensures part-time nurses have either 3 or 4 shifts; (5.24) ensures no nurse works both a morning and afternoon shift in the same day; and (5.28) is the binary constraint.*

- *There are n nurses and 7 days a week, and so there are $7n$ morning shifts (x_{ij}) and $7n$ afternoon shifts (y_{ij}). So altogether there are $14n$ decision variables.*
- *There are 7 of each of constraints 5.22 and 5.23. There are n of each of constrains 5.25, 5.26, and 5.27. There are $7 \times n$ of constraint 5.24. Therefore altogether there are $(2 \times 7) + (3 \times n) + (1 \times 7 \times n) = 10n + 14$ constraints not including the binary constraints.*

Linearity Tricks

Not all linear problems seem linear at first. For example, it is possible to write constraints with $\min$ or $\max$ functions as linear constraints, by introducing extra dummy variables. Take for example:

Minimise:

$$\max(x_1, x_2)$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 6 \\ x_1, x_2 &\geq 0 \end{aligned}$$

We can introduce a dummy variable y. To ensure that y represents the maximum of x_1 and x_2, we restrict it to be greater than or equal to either of them, and minimise it so that it takes on the value of the maximum of these. This is called a *minimax* problem (minimising the maximum of some variables).

Minimise:

$$y$$

subject to

$$\begin{aligned} y &\geq x_1 \\ y &\geq x_2 \\ x_1 + x_2 &\leq 6 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Conversely, we could have a *maximin* problem.

Chapter 6

Transportation Problems

Learning Outcomes:

- Be able to formulate transportation problems as integer linear programming problems;
- Be able to produce feasible but non-optimal basic solutions;
- Be able to produce optimal solutions using the Stepping-Stone Algorithm;
- Be able to perform calculations to deduce solutions to what-of questions based on an optimal solution.

6.1 Introduction

Transportation problems are a special case of linear programming problems. The name 'transportation problem' comes from the fact that the earliest applications were concerned with the efficient means of transporting goods from a set of sources to a set of destinations.

For example, suppose we have three mines, X, Y and Z supplying coal to three power stations A, B and C. The mines can supply the following amount of coal per day: X supplies 400 tonnes; Y supplies 1200 tonnes; and Z supplies 800 tonnes. The power stations require the following amounts per day: A requires 1000 tonnes; B requires 900 tonnes; and C requires 500 tonnes. The transportation costs from mine to power station are as shown below:

	X	Y	Z
A	5	17	11
B	14	18	12
C	10	25	20

The object is to decide how much coal to transport via each route so as to give the minimum overall cost, and be consistent with the conditions of supply and demand.

The integer programming formulation for this problem is as follows: let x_{ij} (for all $i = x, y, z$ and $j = a, b, c$) be the amount of coal transported from mine i to power station j. Then the linear programming problem becomes:

Minimise: $5x_{xa} + 14x_{xb} + 10x_{xc} + 17x_{ya} + 18x_{yb} + 25x_{yc} + 11x_{za} + 12x_{zb} + 20x_{zc}$

subject to

$$
\begin{aligned}
x_{xa} + x_{xb} + x_{xc} &= 400 && (6.1)\\
x_{ya} + x_{yb} + x_{yc} &= 1200 && (6.2)\\
x_{za} + x_{zb} + x_{zc} &= 800 && (6.3)\\
x_{xa} + x_{ya} + x_{za} &= 1000 && (6.4)\\
x_{xb} + x_{yb} + x_{zb} &= 900 && (6.5)\\
x_{xc} + x_{yc} + x_{zc} &= 500 && (6.6)
\end{aligned}
$$

and $x_{ij} \geq 0$ for all $i = x, y, z$ and $j = a, b, c$.

Here constraints 6.1, 6.2, and 6.3 correspond to the supply capacity of mines X, Y and Z respectively; while constraints 6.4, 6.5, and 6.6 correspond to the demand of power stations A, B and C respectively. The objective corresponds to minimising the overall transportation cost.

General formulation

The general formulation is given by letting:

- a_i be the amount of supply at source i;
- b_j be the amount of demand at destination j;
- c_{ij} be the transportation cost between source i and destination j;
- x_{ij} be the amount to be transported from source i to destination j.

Now:

Minimise:

$$\sum_i \sum_j c_{ij} x_{ij}$$

subject to

$$\sum_j x_{ij} = a_i \text{ for all } i$$

$$\sum_i x_{ij} = b_j \text{ for all } j$$

$$x_{ij} \geq 0 \text{ for all } i, j$$

This could of course now be solved using the Simplex method. However, written like this we can make some observations. The first being that all the coefficients of the decision variables in the constraints are 1. The second is that all the constraints are equalities. The third is that one of the constraints is not actually needed: consider the coal mine and power station example, if constraints 6.1 to 6.5 are satisfied, then constraint 6.6 must also be satisfied, as the supply equals the demand. Thus, although this problem has 9 decision variables and 6 constraints, and so would expect to have 3 non-basic and 6 basic variables, in fact we have 4 non-basic and 5 basic variables.

All this means that we can use simpler methods than the Simplex method to solve them. First we will consider how to find an initial feasible solution (but not necessarily optimal), then we will look into how to adapt this solution to find the optimal allocations.

6.2 Finding Feasible (non-optimal) Solutions

To begin solving transportation problems, we will set the problem out as the following table:

	X	Y	Z	
A	x_{xa} 5	x_{ya} 17	x_{za} 11	10
B	x_{xb} 14	x_{yb} 18	x_{zb} 12	9
C	x_{xc} 10	x_{yc} 25	x_{zc} 20	5
	4	12	8	24

The final column gives the demand of the power stations, and so the variables in each row should sum to these numbers. The final row gives the supply of the mines, and so the variables in each column should sum to these. The costs of each route is given in the bottom corner right of each cell.

We will consider two methods of assigning values to the decision variables that will give a feasible solution: the **North West corner method**, and the **minimum cost method**.

The North West Corner Method

Here we allocate as much as possible to the North West corner of the table. Then, depending on whether supply or demand has not been satisfied, we move either South or East. We then continue to allocate as much as possible in this manner, until all demand has been satisfied.

Example 29 *Using the North West corner method, find an initial solution for the transportation problem with three mines, X, Y and Z supplying coal to three power stations A, B and C. The mines can supply the following amount of coal per day: X supplies 400 tonnes; Y supplies 1200 tonnes; and Z supplies 800 tonnes. The power stations require the following amounts per day: A requires 1000 tonnes; B requires 900 tonnes; and C requires 500 tonnes. The transportation costs from mine to power station are as shown below:*

	X	*Y*	*Z*
A	*5*	*17*	*11*
B	*14*	*18*	*12*
C	*10*	*25*	*20*

Solution to Example 29 *We obtain:*

	X	Y	Z	
A	4 (5)	6 (17)	(11)	10
B	(14)	6 (18)	3 (12)	9
C	(10)	(25)	5 (20)	5
	4	12	8	24

- *First assign as much as possible to the North West corner, that is x_{xa}: the most that x can supply is 4 as that satisfies the column total, but leaves the row total unsatisfied.*
- *So we move East to x_{ya}: the most we can assign here is 6 as that satisfies the row total, leaving the column total unsatisfied.*
- *So we move South to x_{yb}: the most we can assign here is 6 as that satisfies the column total, leaving the row total unsatisfied.*
- *So we move East to x_{zb}: the most we can assign here is 3 as that satisfies the row total, but leave the column total unsatisfied.*
- *So we move South to x_{zc}: the most we can assign here is 5 and that satisfies both the column and row totals, so we stop.*

This yields a cost of: $(4 \times 5) + (6 \times 17) + (6 \times 18) + (3 \times 12) + (5 \times 20) = 366$.

This yields a feasible solution, but it may not be of the best quality. It was entirely determined by the arbitrary ordering of the table.

The Minimum Cost Method

Here we allocate as much as possible to the cheapest cell of the table. Then we continue to add to the next cheapest available cells until all demand and supply constraints are satisfied.

Example 30 *Using the minimum cost method, find an initial solution for the transportation problem given in Example 29.*

Solution to Example 30 *We obtain:*

	X	Y	Z	
A	4 (5)	(17)	6 (11)	10
B	(14)	7 (18)	2 (12)	9
C	(10)	5 (25)	(20)	5
	4	12	8	24

- *First assign as much as possible to the cheapest route, that is x_{xa}: the most that x can supply is 4 as that satisfies the column total, but leaves the row total unsatisfied.*
- *So we move to the next cheapest route, x_{xc}: but we cannot assign anything here as the column total is already satisfied.*
- *So we move to the next cheapest route, x_{za}: the most we can assign here is 6 as that satisfies the row total, leaving the column total unsatisfied.*
- *So we move to the next cheapest route, x_{zb}: the most we can assign here is 2 as that satisfies the column total, leaving the row total unsatisfied.*
- *So we move to the next cheapest available route (as x_{xb} is not available), x_{yb}: the most we can assign here is 7 as that satisfies the row total, leaving the column total unsatisfied.*
- *So we move the next next cheapest available route x_{yc}: the most we can assign here is 5, and that satisfies both the column and row totals, so we stop.*

This yields a cost of: $(4 \times 5) + (6 \times 11) + (7 \times 18) + (2 \times 12) + (5 \times 25) = 361$.

In each of the feasible solutions produced by the North West Corner Method and the Minimum Cost Method, there are 5 basic variables (five routes carrying non-zero amounts of coal) and four non-basic variables (four routes carrying zero coal) as expected.

6.3 The Stepping-Stone Algorithm

The Stepping-Stone Algorithm takes an initial feasible solution (produced by either the North West corner method or the minimum cost method) and improves it to produce an optimal solution. It does this though considering each of the non-basic variables and considering what would be the cost of making that variable basic.

Consider the feasible solution given by Example 29:

	X	Y	Z	
A	4 5	6 17	11	10
B	14	6 18	3 12	9
C	10	25	5 20	5
	4	12	8	24

There are four non-basic variables: x_{az}, x_{bx}, x_{cx}, and x_{cy}. What would be the cost of making one of those variables basic?

- First consider x_{bx}: If this were to be made basic then mine X would be supplying too much coal, so we would have to reduce x_{ax}, which means power station A wouldn't receive enough coal, so we would have to increase x_{ay}, which means mine Y would be supplying too much coal, so we would have to decrease x_{by}. This would even out the amount of coal received by power station B, closing the loop. That is:

	X	Y	Z	
A	4 -5	6 +17	11	10
B	+14	6 -18	3 12	9
C	10	25	5 20	5
	4	12	8	24

Therefore to increase x_{bx} by one unit it would cost: $+14 - 5 + 17 - 18 = 8$, and so it would not improve the solution.

- Now consider x_{az}:

	X	Y	Z	
A	4 5	6 -17	+11	10
B	14	6 +18	3 -12	9
C	10	25	5 20	5
	4	12	8	24

Therefore to increase x_{az} by one unit it would cost: $+11 - 12 + 18 - 17 = 0$, and so it wouldn't make any difference to the solution.

- Now consider x_{cx}:

	X	Y	Z	
A	4 -5	6 +17	11	10
B	14	6 -18	8 +12	9
C	+10	25	5 -20	5
	4	12	8	24

Therefore to increase x_{cx} by one unit it would cost: $+10 - 5 + 17 - 18 + 12 - 20 = -4$, and so making x_{cx} would improve the solution.

The most that x_{cx} can increase by is 4, as that is the most that x_{ax} can decrease by. And so we get the new solution by increasing x_{cx}, x_{ay}, x_{bz} by 4, and decreasing x_{ax}, x_{by}, x_{cz} by 4, yielding an improvement in cost of $4 \times -4 = -16$:

	X	Y	Z	
A	5	10 17	11	10
B	14	2 18	7 12	9
C	4 10	25	1 20	5
	4	12	8	24

The Stepping-Stone Algorithm utilises this logic systematically to produce an optimal solution.

Stepping-Stone Algorithm

Steps:

1. Find an initial feasible solution (e.g. using the North West Corner Method or the Minimum Cost Method).
2. For each non-basic variable:
   (a) Draw a closed path, starting at that non-basic variable, though basic variables.
   (b) Consider the net unit cost of making that non-basic variable basic.
3. If all net costs are greater to or equal to zero, the optimal solution has been found. End algorithm.
4. Else, select the non-basic variable that would result in the largest negative net cost.
5. Add to it the most units possible, subtracting/adding the same amount to all variables along the closed loop.
6. Go to 1).

Note that there may be more than one optimal solution, that is two different solutions that give the same minimum cost. E.g. when making a non-basic variable basic has a net cost change of zero.

Example 31 *First use the North West corner method, and then the stepping-stone algorithm, to find the optimal solution to the following transportation problem:*

	D_1	D_2	D_3	D_4	
O_1	3	1	7	4	300
O_2	2	6	5	9	400
O_3	8	3	3	2	500
	250	350	400	200	1200

Solution to Example 31 *Following the North West Corner method we get:*

	D_1	D_2	D_3	D_4	
O_1	250 (3)	50 (1)	(7)	(4)	300
O_2	(2)	300 (6)	100 (5)	(9)	400
O_3	(8)	(3)	300 (3)	200 (2)	500
	250	350	400	200	1200

The first step of the stepping-stone algorithm:

Path	*Cost*
$O_2D_1 \to O_1D_1 \to O_1D_2 \to O_2D_2$	+ 2 - 3 + 1 - 6 = -6
$O_3D_1 \to O_1D_1 \to O_1D_2 \to O_2D_2 \to O_2D_3 \to O_3D_3$	+ 8 - 3 + 1 - 6 + 5 - 3 = 2
$O_3D_2 \to O_2D_2 \to O_2D_3 \to O_3D_3$	+ 3 - 6 + 5 - 3 = -1
$O_1D_3 \to O_2D_3 \to O_2D_2 \to O_1D_2$	+ 7 - 5 + 6 - 1 = 7
$O_1D_4 \to O_3D_4 \to O_3D_3 \to O_2D_3 \to O_2D_3 \to O_1D_2$	+ 4 - 2 + 3 - 5 + 6 - 1 = 5
$O_2D_4 \to O_3D_4 \to O_3D_3 \to O_2D_3$	+ 9 - 2 + 3 - 5 = 5

Therefore increase O_2D_1 as much as possible, that is by 250, giving:

	D_1	D_2	D_3	D_4	
O_1	(3)	300 (1)	(7)	(4)	300
O_2	250 (2)	50 (6)	100 (5)	(9)	400
O_3	(8)	(3)	300 (3)	200 (2)	500
	250	350	400	200	1200

Solution to Example 31 (continuing from p. 113) *The second step of the stepping-stone algorithm:*

Path	*Cost*
$O_1D_1 \to O_1D_2 \to O_2D_2 \to O_2D_1$	*+ 3 - 1 + 6 - 2 = 6*
$O_3D_1 \to O_2D_1 \to O_2D_3 \to O_3D_3$	*+ 8 - 2 + 5 - 3 = 8*
$O_3D_2 \to O_2D_2 \to O_2D_3 \to O_3D_3$	*+ 3 - 6 + 5 - 3 = -1*
$O_1D_3 \to O_2D_3 \to O_2D_2 \to O_1D_2$	*+ 7 - 5 + 6 - 1 = 7*
$O_1D_4 \to O_3D_4 \to O_3D_3 \to O_2D_3 \to O_2D_2 \to O_1D_2$	*+ 4 - 2 + 3 - 5 + 6 - 1 = 5*
$O_2D_4 \to O_3D_4 \to O_3D_3 \to O_2D_3$	*+ 9 - 2 + 3 - 5 = 5*

Therefore increase O_3D_2 as much as possible, that is by 50, giving:

	D_1	D_2	D_3	D_4	
O_1	3	300 1	7	4	300
O_2	250 2	6	150 5	9	400
O_3	8	50 3	250 3	200 2	500
	250	350	400	200	1200

The third step of the stepping-stone algorithm:

Path	*Cost*
$O_1D_1 \to O_1D_2 \to O_3D_2 \to O_3D_3 \to O_2D_3 \to O_2D_1$	*+ 3 - 1 + 3 - 3 + 5 - 2 = 5*
$O_3D_1 \to O_2D_1 \to O_2D_3 \to O_3D_3$	*+ 8 - 2 + 5 - 3 = 8*
$O_2D_2 \to O_2D_3 \to O_3D_3 \to O_3D_2$	*+ 6 - 5 + 3 - 3 = 1*
$O_1D_3 \to O_3D_3 \to O_3D_2 \to O_1D_2$	*+ 7 - 3 + 3 - 1 = 6*
$O_1D_4 \to O_3D_4 \to O_3D_2 \to O_1D_2$	*+ 4 - 2 + 3 - 1 = 4*
$O_2D_4 \to O_3D_4 \to O_3D_3 \to O_2D_3$	*+ 9 - 2 + 3 - 5 = 5*

Therefore there is not more cost reducing moves, and we have found the optimal allocation.

Maximisation Problems

Note that there may be cases where problems take the same structure, but we wish to maximise some profit instead of minimising cost. In these cases we would:

- Replace the *minimum cost method* with a *maximum cost method* to find a basic feasible solution (note this isn't necessary, the minimum cost method will also find a basic feasible solution, but it may be further away from the optimal).
- When carrying out the stepping-stone algorithm, the optimal solution is found if all net costs are less than or equal to zero, and we select the variable with the greatest positive net cost.

6.4 Degeneration in Transportation Problems

When using the Simplex method to solve linear programming problems in Chapter 5, it was possible for a problem to be *degenerate*. This happens when more than two constraints meet at the optimal solution, and is indicated when a basic variable is equal to zero. This may also happen quite frequently in transportation problems, and it can be recognised easily once an initial solution has been produced.

In general, given that there are m sources and n destinations, there will be $m + n - 1$ basic variables. Once an initial solution has been found (for example using the North West Corner method or the Minimum Cost method), we would expect to see $m+n-1$ cells of the table filled with non-zero values. If there are less of these, then the problem is *degenerate*.

Consider the following example where the initial solution was allocated by the minimum cost method:

	W	X	Y	Z	
A	9	1 (8)	9 (3)	7	10
B	4 (2)	3 (5)	4	8	7
C	3	2	6	3 (1)	3
	4	4	9	3	20

Here we would expect $m + n - 1 = 4 + 3 - 1 = 6$ basic variables, however there are only 5 non-zero basic variables, and so the problem is degenerate. This causes problems, for example as we cannot perform the stepping-stone algorithm now, as it is not possible to draw closed paths from basic variable to basic variable every time. Try it above!

This can be overcome by identifying which of the zero-variables is the basic variable, and labelling it as 0. Then the stepping-stone algorithm can use this cell to create closed paths. However, identifying this cell cannot be done at this stage, it has to be done *while* creating the initial feasible solution:

- Choose a method to find the initial feasible solution;
- When filling in the cells, if they satisfy *either* the row *or* column totals, tick that row or column;
- When filling in the cells, if they satisfy **both** row and column totals, *only* tick either the row or column;
- At the end, either all rows and columns are ticked (no degeneration), or there is *one row* and *one column* left unticked;
- If there are any rows and columns that are not ticked, but are satisfied, then place *one zero* in the cell that would tick these off.

Consider the following example:

Example 32 *Find an initial feasible solution for the following degenerate problem using the North West Corner method:*

	W	X	Y	Z	
A	9	8	3	7	10
B	2	5	4	8	7
C	3	2	6	1	3
	4	4	9	3	20

Use the stepping-stone algorithm to find the optimal solution.

Solution to Example 32 *Following the North West Corner method we get:*

- *Set* $x_{AW} = 4$ *- tick off column W;*
- *Set* $x_{AX} = 4$ *- tick off column X;*
- *Set* $x_{AY} = 2$ *- tick off row A;*
- *Set* $x_{BY} = 7$ *- we can either tick off column Y or row B - tick off column Y;*
- *Set* $x_{CZ} = 3$ *- we can either tick off column Z or row C - tick off row C;*
- *That satisfies all row and column totals, but there are two left unticked: row B and column Z; so place a 0 in cell BZ.*

Giving:

	W	X	Y	Z	
A	4 (9)	4 (8)	2 (3)	(7)	10
B	(2)	(5)	7 (4)	0 (8)	7
C	(3)	(2)	(6)	3 (1)	3
	4	4	9	3	20

The first step of the stepping-stone algorithm:

Path	*Cost*
$AZ \to BZ \to BY \to AY$	*Not possible.*
$BW \to AW \to AY \to BY$	+ 2 - 9 + 3 - 4 = -8
$BX \to AX \to AY \to BY$	+ 5 - 8 + 3 - 4 = -4
$CW \to AW \to AY \to BY \to BZ \to CZ$	+ 3 - 9 + 3 - 4 + 8 - 1 = 0
$CX \to AX \to AY \to BY \to BZ \to CZ$	+ 2 - 8 + 3 - 4 + 8 - 1 = 0
$CY \to BY \to BZ \to CZ$	+ 6 - 4 + 8 - 1 = 9

Solution to Example 32 (continuing from p. 117) *Therefore increase BW as much as possible, giving:*

	W	X	Y	Z	
A	9	4 8	6 3	7	10
B	4 2	5	3 4	0 8	7
C	3	2	6	3 1	3
	4	4	9	3	20

The second step of the stepping-stone algorithm:

Path	*Cost*
$AW \to AY \to BY \to BW$	+ 9 - 3 + 4 - 2 = 8
$AZ \to BZ \to BY \to AY$	Not possible.
$BX \to AX \to AY \to BY$	+ 5 - 8 + 3 - 4 = -4
$CW \to BW \to BZ \to CZ$	+ 3 - 2 + 8 - 1 = 8
$CX \to AX \to AY \to BY \to BZ \to CZ$	+ 2 - 8 + 3 - 4 + 8 - 1 = 0
$CY \to BY \to BZ \to CZ$	+ 6 - 4 + 8 - 1 = 9

Therefore increase BX as much as possible, giving:

	W	X	Y	Z	
A	9	1 8	9 3	7	10
B	4 2	3 5	4	0 8	7
C	3	2	6	3 1	3
	4	4	9	3	20

Solution to Example 32 (continuing from p. 118) *The third step of the stepping-stone algorithm:*

Path	*Cost*
$AW \to AX \to BX \to BW$	$+ 9 - 8 + 5 - 2 = 4$
$AZ \to BZ \to BX \to AX$	*Not possible.*
$BY \to BX \to AX \to AY$	$+ 4 - 5 + 8 - 3 = 4$
$CW \to BW \to BZ \to CZ$	$+ 3 - 2 + 8 - 1 = 8$
$CX \to BX \to BZ \to CZ$	$+ 2 - 5 + 8 - 1 = 4$
$CY \to CZ \to BZ \to BX \to AX \to AY$	$+ 6 - 1 + 8 - 5 + 8 - 3 = 13$

Therefore there is not more cost reducing moves, and we have found the optimal allocation.

6.5 Unequal Supply and Demand

In all cases considered previously, the supply and the demand have been the same. This made our algorithm work nicely so that we could use simple methods like the North West Corner method to find basic feasible solutions, that were tight to all constraints. However, in most practical situations this will not be the case.

We deal with unequal supply and demand by creating a dummy supplier (if demand is greater than supply), or a dummy destination (if supply is greater than demand). Costs to or from dummy suppliers and dummy destinations will be set to zero. However, these should not be considered when creating a basic feasible solution using the minimum cost method, as we would end up filling the dummy rows and columns first, which are not very helpful.

Let's do an example:

Example 33 *Find an feasible solution for the following problem using the minimum cost method:*

	X	Y	Z	
A	1	5	5	7
B	4	2	2	4
C	3	6	1	3
	5	3	4	

Use the stepping-stone algorithm to find the optimal solution.

Solution to Example 33 *In this case the total demand (columns) is equal to 12, but the total supply (rows) is equal to 14. Therefore we need to add a dummy demand column to receive the excess 2 units of supply:*

	X	Y	Z	*Dummy*	
A	$_{1}$	$_{5}$	$_{5}$	$_{0}$	*7*
B	$_{4}$	$_{2}$	$_{2}$	$_{0}$	*4*
C	$_{3}$	$_{6}$	$_{1}$	$_{0}$	*3*
	5	*3*	*4*	*2*	*14*

Then using the minimum cost method (note that this is degenerate, and so a zero valued basic variable is added):

	X	Y	Z	*Dummy*	
A	*5* $_{1}$	$_{5}$	$_{5}$	*2* $_{0}$	*7*
B	$_{4}$	*3* $_{2}$	*1* $_{2}$	*0* $_{0}$	*4*
C	$_{3}$	$_{6}$	*3* $_{1}$	$_{0}$	*3*
	5	*3*	*4*	*2*	*14*

The first step of the stepping-stone algorithm:

Path	*Cost*
$AY \to AD \to BD \to BY$	$+ 5 - 0 + 0 - 2 = 3$
$AZ \to AD \to BD \to BZ$	$+ 5 - 0 + 0 - 2 = 3$
$BX \to AX \to AD \to BD$	*Not possible.*
$CX \to AX \to AD \to BD \to BZ \to CZ$	*Not possible.*
$CY \to BY \to BZ \to CZ$	$+ 6 - 2 + 2 - 1 = 5$
$CD \to CZ \to BZ \to BD \to AX \to AY$	*Not possible.*

Therefore there is not more cost reducing moves, and we have found the optimal allocation.

6.6 Routes Becoming Unavailable

Consider that an optimal solution has been found for a a given transportation problem. Now, one of the used routes becomes unavailable (for example due to a natural disaster, or political turmoil). How can we adapt the found solution to account for this?

The impossible routes are dealt with by assuming that the cost of transportation by that route becomes very high. This cost is usually denoted by M. Then we continue the stepping-stone algorithm, looking for moves with costs with greatest negative multiples of M.

Example 34 *Consider that the following optimal allocation has been found for a transportation problem with five suppliers and three destinations:*

	V	W	X	Y	Z	
A	$_{5}$	$_{7}$	$_{10}$	1_{5}	4_{3}	5
B	3_{8}	3_{6}	4_{9}	$_{12}$	$_{14}$	10
C	$_{10}$	$_{9}$	6_{8}	4_{10}	$_{15}$	10
	3	3	10	5	4	25

The route BX becomes unavailable. Find a new optimal solution.

Solution to Example 34 *We assign the cost of route BX an unspecified very high cost M:*

	V	W	X	Y	Z	
A	$_{5}$	$_{7}$	$_{10}$	1_{5}	4_{3}	5
B	3_{8}	3_{6}	4_{M}	$_{12}$	$_{14}$	10
C	$_{10}$	$_{9}$	6_{8}	4_{10}	$_{15}$	10
	3	3	10	5	4	25

Solution to Example 34 (continuing from p. 121) *We continue the stepping-stone algorithm:*

Path	*Cost*
$AV \to AY \to CY \to CX \to BX \to BV$	$+ 5 - 5 + 10 - 8 + M - 8 = M - 6$
$AW \to AY \to CY \to CX \to BX \to BW$	$+ 7 - 5 + 10 - 8 + M - 6 = M - 2$
$AX \to AY \to CY \to CX$	$+ 10 - 5 + 10 - 8 = 7$
$BY \to CY \to CX \to BX$	$+ 12 - 10 + 8 - M = 10 - M$
$BZ \to BX \to CX \to CY \to AY \to AZ$	$+ 14 - M + 8 - 10 + 5 - 3 = 14 - M$
$CV \to BV \to BX \to CX$	$+ 10 - 8 + M - 8 = 10 + M$
$CW \to BW \to BX \to CX$	$+ 9 - 6 + M - 8 = M - 5$
$CZ \to CY \to AY \to AZ$	$+ 15 - 10 + 5 - 3 = 7$

There are two options to reduce the flow in BX, either increasing BY by 4, costing a total of $40 - 4M$*, or increasing BZ by 4, costing a total of* $56 - 4M$*. Therefore increasing BY has the lowest net cost. Giving:*

	V	W	X	Y	Z	
A	$_{5}$	$_{7}$	$_{10}$	1_{5}	4_{3}	5
B	3_{8}	3_{6}	$_{M}$	4_{12}	$_{14}$	10
C	$_{10}$	$_{9}$	10_{8}	0_{10}	$_{15}$	10
	3	3	10	5	4	25

Notice this results in more than one cell reducing to zero, but we need to keep one to deal with degeneracy. It is useful to set CY as a zero basic variable, as BX is not to be used any more. Now, continuing the stepping-stone algorithm:

Path	*Cost*
$AV \to AY \to BY \to BV$	$+ 5 - 1 + 12 - 8 = 8$
$AW \to AY \to BY \to BW$	$+ 7 - 1 + 12 - 6 = 12$
$AX \to AY \to CY \to CX$	$+ 10 - 1 + 10 - 8 = 11$
BX	*We do not want to increase this.*
$BZ \to BY \to AY \to AZ$	$+ 14 - 12 + 1 - 3 = 0$
$CV \to BV \to BY \to CY$	*Not possible.*
$CW \to BW \to BY \to CY$	*Not possible.*
$CZ \to CY \to AY \to AZ$	*Not possible.*

And so we know this allocation is now optimal.

6.7 Sensitivity Analysis

Sensitivity analysis is a way of determining the effect of changing some parameters on the optimal solution. For example, we may want to know what the effect of changing some of the transportation costs. In particular, we can find out the range of values particular route costs can take before the optimal solution changes.

The technique is different for when we consider basic and non-basic variables:

- For **non-basic variables**:
  Set the cost to be P. Draw the loop from itself through basic variables, consider the net cost of the loop in terms of P, determine the values of P for which the net cost remains positive.

- For **basic variables**:
  Set the cost to be P. Consider *all* non-basic variables for which their loop contains the considered basic variable. Consider the net cost of the loops in terms of P, determine the values of P for which the net cost remains positive.

Example 35 *Consider the following optimal allocation:*

	D_1	D_2	D_3	D_4	
O_1	3	300 (1)	7	4	300
O_2	250 (2)	6	150 (5)	9	400
O_3	8	50 (3)	250 (3)	200 (2)	500
	250	350	400	200	1200

(i) For what range of values can the cost of O_1D_3 take before the optimal allocation changes?
(ii) For what range of values can the cost of O_2D_4 take before the optimal allocation changes?
(iii) For what range of values can the cost of O_2D_3 take before the optimal allocation changes?

Solution to Example 35 *In turn:*

(i) O_1D_3 *is a non-basic variable. Set its cost to* P*. We consider it's loop:*
$O_1D_3 \to O_3D_3 \to O_3D_2 \to O_1D_2$
with net cost: $P - 3 + 3 - 1 = P - 1$.
The allocation remains optimal if this net cost is positive, so $P - 1 > 0$*, so* $P > 1$.

(ii) O_2D_4 *is a non-basic variable. Set its cost to* P*. We consider it's loop:*
$O_2D_4 \to O_3D_4 \to O_3D_3 \to O_2D_3$
with net cost: $P - 2 + 3 - 5 = P - 4$.
The allocation remains optimal if this net cost is positive, so $P - 4 > 0$*, so* $P > 4$.

(iii) O_2D_3 *is a basic variable. Set its cost to be* P*. Consider all non-basic variables whose loops contain* O_2D_3*:*

- $O_1D_1 \to O_1D_2 \to O_3D_2 \to O_3D_3 \to O_2D_3 \to O_2D_1$
  with net cost: $3 - 1 + 3 - 3 + P - 2 = P$ *For this allocation remains optimal this net cost needs to be positive, so* $P > 0$.
- $O_2D_2 \to O_2D_3 \to O_3D_3 \to O_3D_2$
  with net cost: $6 - P + 3 - 3 = 6 - P$ *For this allocation remains optimal this net cost needs to be positive, so* $6 - P > 0$*, so* $P < 6$.
- $O_2D_4 \to O_3D_4 \to O_3D_3 \to O_2D_3$
  with net cost: $9 - 2 + 3 - P = 10 - P$ *For this allocation remains optimal this net cost needs to be positive, so* $10 - P > 0$*, so* $P < 10$.
- $O_3D_1 \to O_2D_1 \to O_2D_3 \to O_3D_3$
  with net cost: $8 - 2 + P - 3 = 3 + P$ *For this allocation remains optimal this net cost needs to be positive, so* $3 + P > 0$*, so* $P > -3$.

All four inequalities need to be satisfied in order for the allocation to remain optimal. That is $P > 0$ *and* $P < 6$ *and* $P < 10$ *and* $P > -3$*, and so* $0 < P < 6$.

Chapter 7

Dynamic Programming

Learning Outcomes:

- Be able to formulate problems as minimisation or maximisation problems on directed acyclic graphs;
- Be able to understand Bellman's Optimality Principle;
- Be able to solve problems on directed acyclic graphs using value iteration.

7.1 Introduction

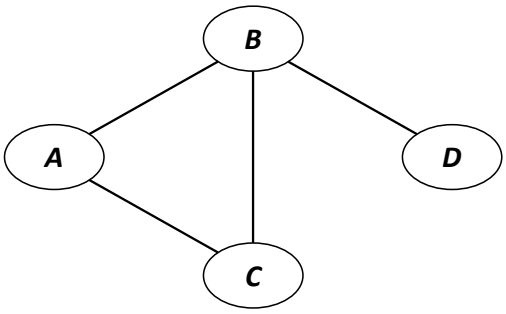
Dynamic programming is another operational research technique in which one attempts to find the best solution, that is the optimal solution from many possible solutions. It is usual to think of these types of problems as problems on networks, for example shortest path problems. However, other problems that are not immediately thought of as network problems can be formulated as such, can be solved in the same manner.

7.2 Directed Acyclic Graphs

It will be useful to recall some concepts on graphs, and to recognise directed acyclic graphs (DAGs) which are central to the study of dynamic programming. A graph $G = (V, E)$ is a mathematical object than consists of two sets:

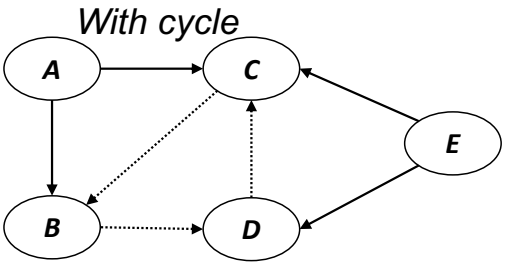
- V: a set of vertices, and
- E: a set of edges. An edge is a pair of vertices $e = (u, v) \in E$, such that $u, v \in V$.

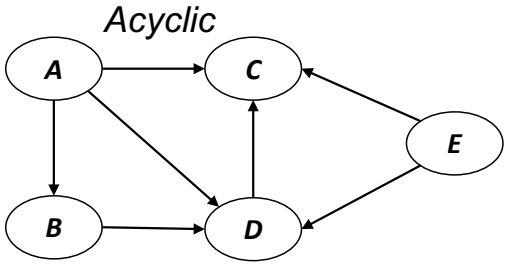
A graph can be drawn, as a network. They are useful for storing information about the relationships between things. For example family trees or road networks. As an example, let $G = (V, E)$ with $V = \{A, B, C, D\}$ and $E = \{(A, B), (A, C), (B, C), (B, D)\}$. This is drawn as:



If the edges are defined as *ordered pairs*, then the graph is called *directed*, and is drawn with arrows. Each edge can also have a weight, a numerical value associated with it, for example it's length, cost, reward, etc.

A *path* is a set of connected edges. For example, the path A-C-D exists if $\{(A, C), (C, D)\} \subseteq E$. A *cycle* is a path that begins and ends at the same node. If a graph contains **no cycles** then it is called *acyclic*. Consider the two directed graphs below, one contains a cycle (highlighted with dotted edges), and another is acyclic.





7.3 Bellman's Optimality Principle

Dynamic programming centres on Bellman's Principle of Optimality:

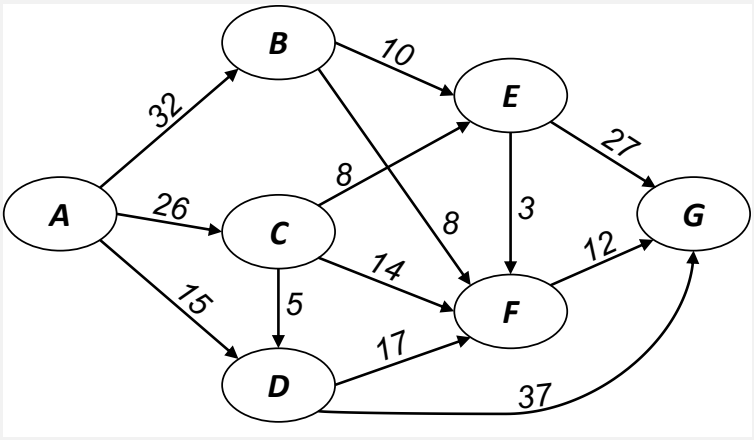
Bellman's Principle of Optimality

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

When thinking about a solution as a set of steps to follow, it can be interpreted as: if the optimal solution when starting at step X contains step Y, then the steps following step Y are the same the optimal solution if we had started at step Y. Therefore it is useful to formulate these problems as routing problems on an acyclic network, and think of these steps as the vertices to visit along that route.

Let's use a shortest path problem as an example:

Example 36 *Consider the following acyclic network:*



What is the shortest path to take between node A and node G?

Bellman's Principle of Optimality states that, for example:

- *if* the shortest path from D to G is D-E-G,
- *and if* the shortest path from A to G contains D,
- *then* the shortest path from A to G contains D-E-G.

7.4 Value Iteration

Value iteration is a technique that allows us to work backwards from G to A, finding the shortest path from each node to G each time, by incorporating the previously found shortest paths. We will do this systematically, by defining the following, let:

- V be the set of vertices or nodes in the graph,
- E be the set of all directed edges in the graph,
- $A_v = \{u \in V \mid (v, u) \in E\}$, be the set of all vertices u such that there exists an edge from v to u,
- r_{vu} be the extra distance or cost incurred by choosing the edge (v, u),
- f_v be the shortest distance from v to the destination.

We can then work backwards through the vertices filling these in as we go, where

$$f_v = \min_{u \in A_v} (r_{vu} + f_u)$$

We set $r_{vu} = \infty$ if the path doesn't exist, but in practice we just ignore these. In order to be able to work backwards from the destination to the origin, we need to order the nodes by maximum number of steps to the destination (as the graphs are acyclic, this ensures that each node has a definite, finite, maximum number of steps to the destination).

Solution to Example 36 *First, order the nodes by maximum number of steps to the destination:*

$$G - F - E - B - D - C - A$$

Now we iteratively work through these vertices, finding f_v each time:

v	u	r_{vu}	$r_{vu} + f_u$	f_v
G	-	0	0	0
F	G	12	$12 + 0 = 12$	12
E	G	27	$27 + 0 = 27$	
	F	3	$3 + 12 = 15$	15
B	E	10	$10 + 15 = 25$	
	F	8	$8 + 12 = 20$	20
D	F	17	$17 + 12 = 29$	
	G	37	$37 + 0 = 37$	29
C	E	8	$8 + 15 = 23$	
	F	14	$14 + 12 = 26$	
	D	5	$5 + 29 = 34$	23
A	B	32	$32 + 20 = 52$	
	C	26	$26 + 23 = 49$	
	D	15	$15 + 29 = 44$	44

Therefore the cost of the shortest path is 44. This was obtained by:

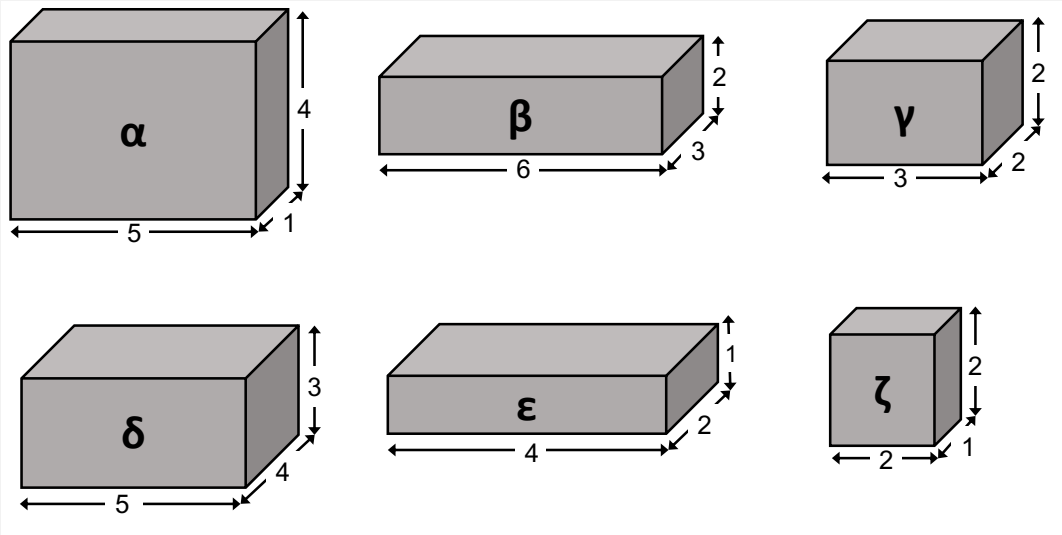
- $44 = 15 + 29$, *going from A to D,*
- $29 = 17 + 12$, *going from D to F,*
- $12 = 12 + 0$, *going from F to G.*

So the shortest path is A-D-F-G with cost 44.

Note: you may have heard of an algorithm called Dijkstra's algorithm for solving shortest path problems. This is a special case of dynamic programming where the problem is symmetric, that is the shortest route from A to B must be the shortest route from B to A. This isn't always the case with some problems.

Some problems that at first seem not to have anything to do with directed acyclic graphs can be solved in this way. Consider the maximisation example below:

Example 37 *Consider the six boxes below:*

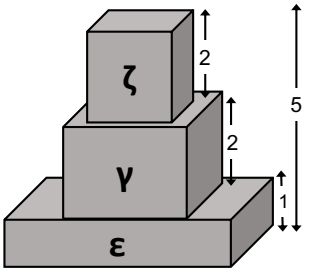


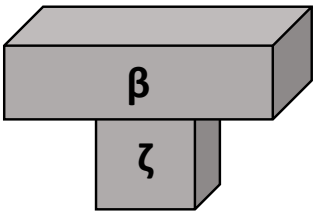
Their dimensions are given by:

Box	*Length*	*Width*	*Height*
α	*5*	*1*	*4*
β	*6*	*3*	*2*
γ	*3*	*2*	*2*
δ	*5*	*4*	*3*
ϵ	*4*	*2*	*1*
ζ	*2*	*1*	*2*

The boxes' contents are fragile and must remain the correct way up. A boxes can only be stacked on top of another box if the its length is less the length of the box below, and if its width is less than the width of the box below.
What is the tallest stack of boxes than can be constructed?

For clarity, consider the examples of stacks below. On the left is ϵ-γ-ζ, producing a stack of height 5, and on the right is an example of a stack that is not allowed, ζ-β, as the length of β is greater than the length of ζ:





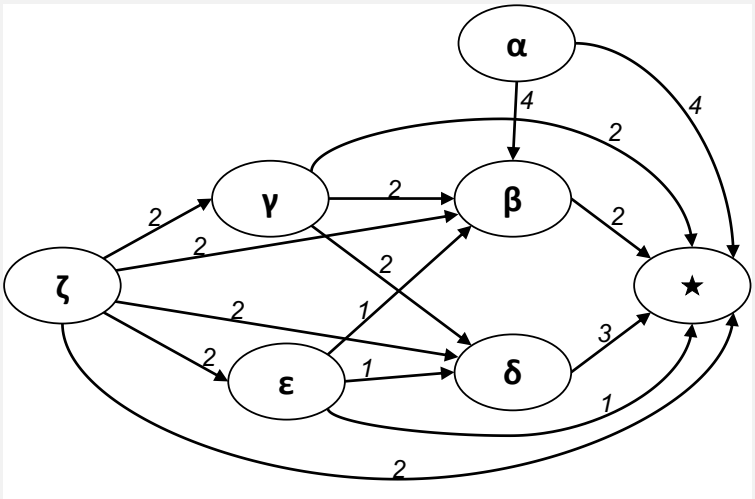
Though this doesn't seem that this is a network problem, we can model this as a directed acyclic graph. There is a clear ordered relationship between each pair of boxes: whether they can be stacked on top of each other or not. We will make use of this in our solution.

Note now that this is a maximisation problem, and so the value iteration equation now becomes

$$f_v = \max_{u \in A_v} (r_{vu} + f_u)$$

We are ready to solve the problem:

Solution to Example 37 *Draw the problem as a directed acyclic graph: there is a directed edge from box x to box y if x can be placed on top of y. The edges (x, y) have weight equal to the height of x. We also need to include the dummy node representing the surface in which they stack upon:*



Solution to Example 37 (continuing from p. 130) *Ordering the boxes by number of steps to the destination gives:*

$$\star - \beta - \delta - \gamma - \epsilon - \alpha - \zeta$$

And perform value iteration:

v	u	r_{vu}	$r_{vu} + f_u$	f_v
$\star$	-	0	0	0
β	$\star$	2	$2 + 0 = 2$	2
δ	$\star$	3	$3 + 0 = 3$	3
γ	$\star$	2	$2 + 0 = 2$	
	β	2	$2 + 2 = 4$	
	δ	2	$2 + 3 = 5$	5
ϵ	β	1	$1 + 2 = 3$	
	δ	1	$1 + 3 = 4$	
	$\star$	1	$1 + 0 = 1$	4
α	$\star$	4	$4 + 0 = 4$	
	β	4	$4 + 2 = 6$	6
ζ	γ	2	$2 + 5 = 7$	
	β	2	$2 + 2 = 4$	
	δ	2	$2 + 3 = 5$	
	ϵ	2	$2 + 4 = 6$	
	$\star$	2	$2 + 0 = 2$	7

Therefore the height of the tallest stack is 7. This was obtained by:

- $7 = 2 + 5$, *stacking* ζ *on top of* γ,
- $5 = 2 + 3$, *stacking* γ *on top of* δ,
- $3 = 3 + 0$, *stacking* δ *on top of the surface.*

So the tallest stack is δ-γ-ζ *with a height of 7.*

Note here it was not possible to rotate the boxes. Think how we might adapt this problem to allow for box rotations. One way would be to treat rotated versions of the boxes as separate boxes (that is, as distinct vertices in the graph), and ensure no edges are drawn between a box and a rotated version of itself.

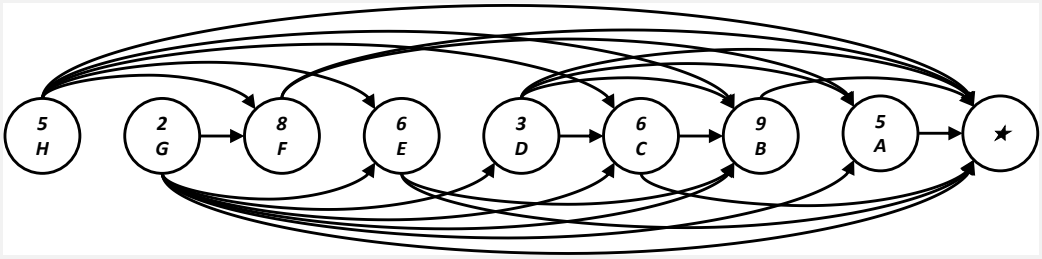
Another example:

Example 38 *Consider the following sequence of numbers:*

$$5 - 2 - 8 - 6 - 3 - 6 - 9 - 5$$

What is its longest subsequence of increasing numbers?

Solution to Example 38 *Order is important here: it can be drawn as a directed acyclic graph. Let each number be a vertex, there's an edge between two vertices if one number is greater than the other, and behind the other in the sequence, with a dummy vertex to end the sequence. The DAG is:*



The weight of each edge is 1, denoting that including that number increases the length of the subsequence by 1. We perform value iteration (utilising the sequence's natural order):

v	u	r_{vu}	$r_{vu} + f_u$	f_v
$\star$	-	0	$0 + 0 = 0$	0
A5	$\star$	1	$1 + 0 = 1$	1
B9	$\star$	1	$1 + 0 = 1$	1
C6	$\star$	1	$1 + 0 = 1$	
	B9	1	$1 + 1 = 2$	2
D3	$\star$	1	$1 + 0 = 1$	
	A5	1	$1 + 1 = 2$	
	B9	1	$1 + 1 = 2$	
	C6	1	$1 + 2 = 3$	3
E6	$\star$	1	$1 + 0 = 1$	
	B9	1	$1 + 1 = 2$	2
F8	$\star$	1	$1 + 0 = 1$	
	B9	1	$1 + 1 = 2$	2
G2	$\star$	1	$1 + 0 = 1$	
	A5	1	$1 + 1 = 2$	
	B9	1	$1 + 1 = 2$	
	C6	1	$1 + 2 = 3$	
	D3	1	$1 + 3 = 4$	
	E6	1	$1 + 2 = 3$	
	F8	1	$1 + 2 = 3$	4
H6	$\star$	1	$1 + 0 = 1$	
	B9	1	$1 + 1 = 2$	
	C6	1	$1 + 2 = 3$	
	E6	1	$1 + 2 = 3$	
	F8	1	$1 + 2 = 3$	3

Therefore the length of the longest increasing subsequence is 4, obtained by $2 - 3 - 6 - 9$.

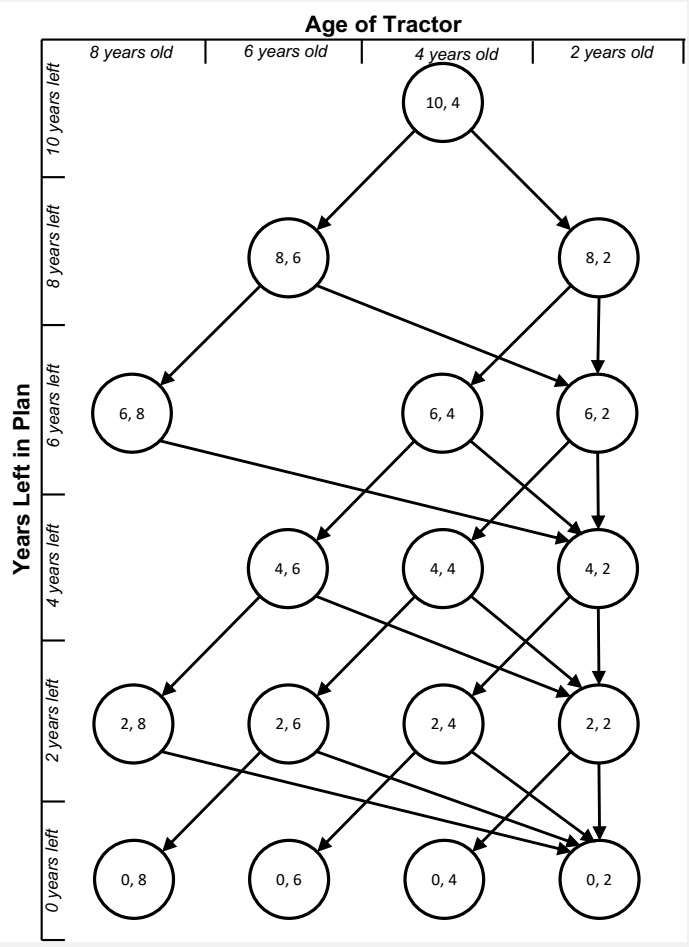
We've seen that dynamic programming is useful in problems that have an inherent order to them. This works well with time-based problems, as discrete time points have an order.

Example 39 *A small farm in Ceredigion has a tractor. Once the tractor reaches 8 years old it must be replaced, costing £30k. Every 2 years the tractor is inspected and will either be repaired at a cost, or traded in and replaced. The trade-in value and cost of repair depends on its age:*

Age (years)	*2*	*4*	*6*	*8*
Repair cost	*£8k*	*£4k*	*£10k*	*-*
Trade-in value	*£15k*	*£10k*	*£3k*	*£0*

The current tractor is now 4 years old. Devise a plan for the next 10 years.

Solution to Example 39 *The important step here is to identify what the states are. Let n the be number of years left in the plan, and let i be the age of the tractor. So the states are the pair (n, i), drawn systematically:*

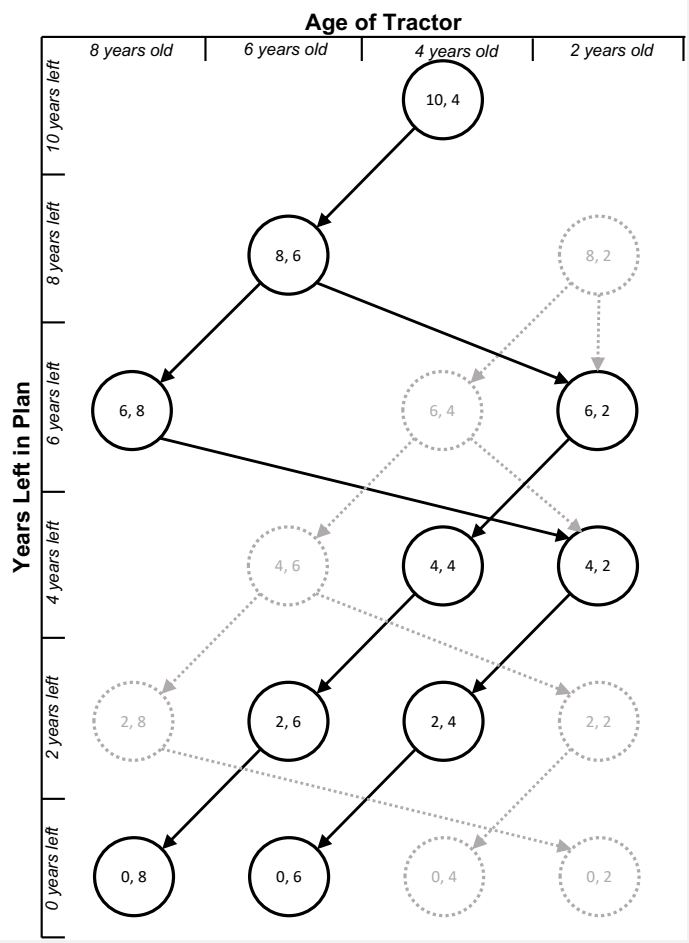


Solution to Example 39 (continuing from p. 133) *Looking at this graph, all arrows pointing to $i = 2$ correspond to a replacement (R), and all other arrows correspond to a repair (F). So we can perform value iteration:*

v	Action	u	r_{vu}	$r_{vu} + f_u$	f_v
(0, 2)	-	-	-15	-15 + 0 = -15	-15
(0, 4)	-	-	-10	-10 + 0 = -10	-10
(0, 6)	-	-	-3	-3 + 0 = -3	-3
(0, 8)	-	-	0	0 + 0 = 0	0
(2, 2)	R	(0, 2)	15	15 - 15 = 0	
	F	(0, 4)	8	8 - 10 = -2	-2
(2, 4)	R	(0, 2)	20	20 - 15 = 5	
	F	(0, 6)	4	4 - 3 = 1	1
(2, 6)	R	(0, 2)	27	27 - 15 = 12	
	F	(0, 8)	10	10 + 0 = 10	10
(2, 8)	R	(0, 2)	30	30 - 15 = 15	15
(4, 2)	R	(2, 2)	15	15 - 2 = 13	
	F	(2, 4)	8	8 + 1 = 9	9
(4, 4)	R	(2, 2)	20	20 - 2 = 18	
	F	(2, 6)	4	4 + 10 = 14	14
(4, 6)	R	(2, 2)	27	27 - 2 = 25	
	F	(2, 8)	10	10 + 15 = 25	25
(6, 2)	R	(4, 2)	15	15 + 9 = 24	
	F	(4, 4)	8	8 + 14 = 22	22
(6, 4)	R	(4, 2)	20	20 + 9 = 29	
	F	(4, 6)	4	4 + 25 = 29	29
(6, 8)	R	(4, 2)	30	30 + 9 = 39	39
(8, 2)	R	(6, 2)	15	15 + 22 = 37	
	F	(6, 4)	8	8 + 29 = 37	37
(8, 6)	R	(6, 2)	27	27 + 22 = 49	
	F	(6, 8)	10	10 + 39 = 49	49
(10, 4)	R	(8, 2)	20	20 + 37 = 57	
	F	(8, 6)	4	4 + 49 = 53	53

The results here are interesting, as there are a number of states where the two decisions give the same f_v value, meaning it doesn't matter what action you should take here. Interpreting this can be difficult. We can help ourselves by redrawing the original directed acyclic graph, but only leaving in the optimal actions. We also shade out any states and subsequent actions that cannot be reached from the starting state (10, 4) by taking optimal actions.

Solution to Example 39 (continuing from p. 134) *The resulting graph is:*



Therefore there are two optimal plans:

- *(10, 6) - F - (8, 6) - R - (6, 2) - F - (4, 4) - F - (2, 6) - F - (0, 8)*
- *(10, 6) - F - (8, 6) - F - (6, 8) - R - (4, 2) - F - (2, 4) - F - (0, 6)*

That is, either replace after 2 years and fix every other time, or replace after 4 years and fix every other time.

Chapter 8

Project Management

Learning Outcomes:

- Be able to draw critical path diagrams for project management problems;
- Be able to identify critical paths for a project;
- Be able to determine allowable delays using floats;
- Be able to find shortest possible project durations with minimum cost.

8.1 Introduction

Critical paths are an operational research technique that can determine the minimum duration of a project that contains many sub-tasks. Some of these sub-tasks or activities may be carried out in parallel, but some may need to be sequential. Consider that a project manager has provided the following information:

- The numbers or titles of each **activity**.
- The **duration** of each activity.
- A table of **precedence** or pre-requisites, stating which tasks need to have been completed before others can begin.

8.2 Activity on Nodes Diagrams

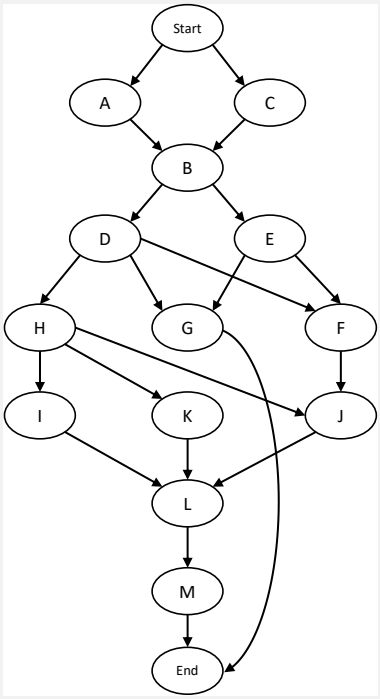
This information can be displayed in an **activity on nodes diagram**, which can be analysed. Consider an example:

Example 40 *Consider the project of relocating an existing factory to a more convenient site. As well as plans for the new site this involves the planned closure of the existing factory, purchase of new machines, movement of existing equipment, etc. The managers provide the following table, with durations in weeks:*

Number	*Activity*	*Duration*	*Precedence*
A	*Select new site*	*12*	
B	*Obtain permission and grant*	*26*	*A, C*
C	*Select architect*	*2*	
D	*Prepare plans*	*4*	*B*
E	*Select builder*	*1*	*B*
F	*Build factory*	*50*	*D, E*
G	*Phase out old factory*	*8*	*D, E*
H	*Plan layout of new factory*	*2*	*D*
I	*Obtain new machines*	*45*	*H*
J	*Move old machines*	*2*	*F, H*
K	*Recruit new staff*	*3*	*H*
L	*Train new staff*	*6*	*I, J, K*
M	*Commission new factory*	*2*	*L*

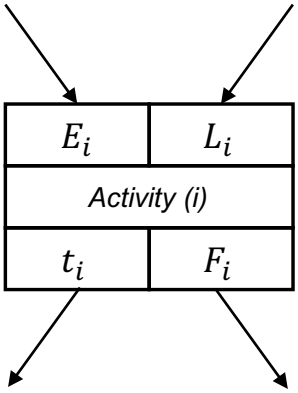
Draw this project as a network of pre-requisites.

Solution to Example 40 *The network would be:*



However this network does not contain a number of key information, namely the durations. We can also determine information such as the earliest staring time, latest starting time, and float. A better diagram would contain this information, and such a diagram is called an activity on nodes diagram.

Each node would look like:



Here:

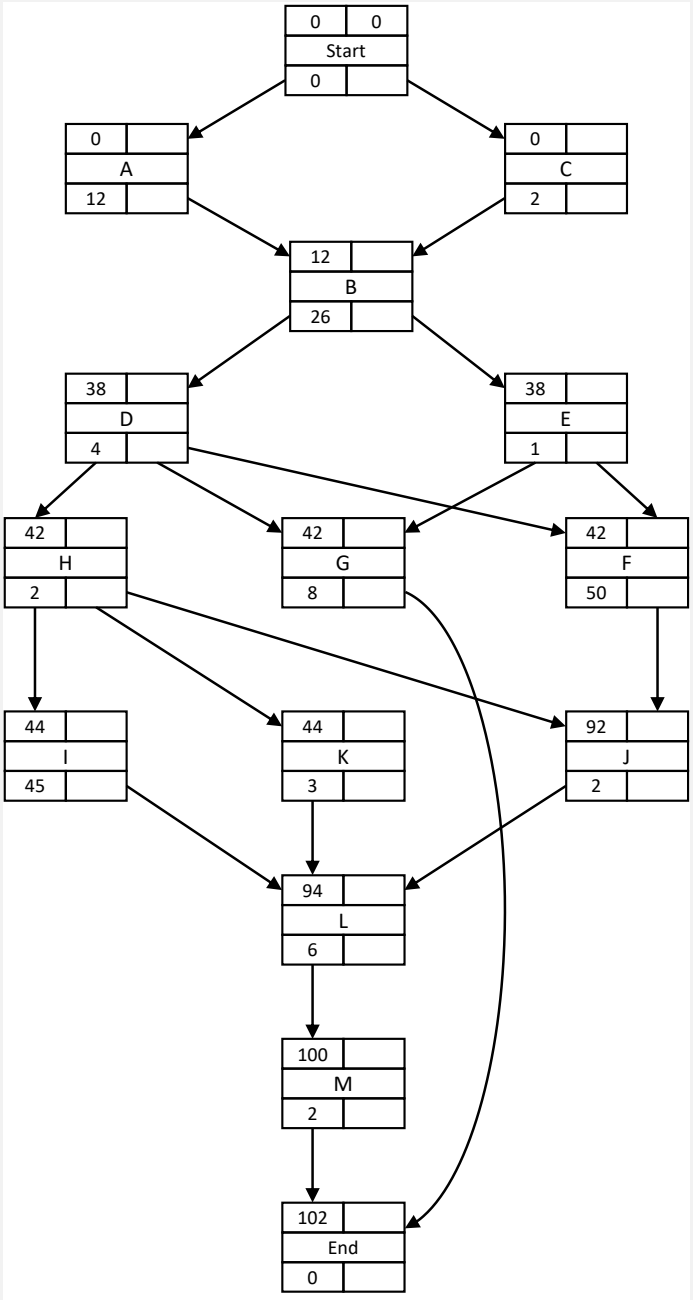
- i is the activity number.
  The activity's number is noted in the middle of the node.
- t_i is the **duration** of activity i.
  This information is provided by the project managers.
- E_i is the **earliest starting time** of activity i.
  Where $E_i = \max_{k \in K_i}(E_k + t_k)$, and K_i is the set of activities directly preceding i.
- L_i is the **latest starting time** of activity i.
  Where $L_i = \min_{m \in M_i}(L_m - t_i)$, and M_i is the set of activities directly following i.
- F_i is the **float** of activity i. That is the amount of slack time for each activity, or the maximum amount of time the activity can be delayed without affecting other activities. $F_i = L_i - E_i$.

To calculate all E_i, L_i and F_i we first need to draw the activity on nodes diagram filling in the activity number and duration only, leaving the other boxes empty. The *start* activity has a duration of 0 and $E_{\text{start}} = L_{\text{start}} = 0$, and the *end* activity has a duration of 0. Then we do a **forward pass**, starting from the *start* activity and filling in the E_is until we reach the *end*. Then we set $L_{\text{end}} = E_{\text{end}}$ and do a **backward pass**, starting at the *end* and filling in each of the L_is and F_is until we reach the *start*.

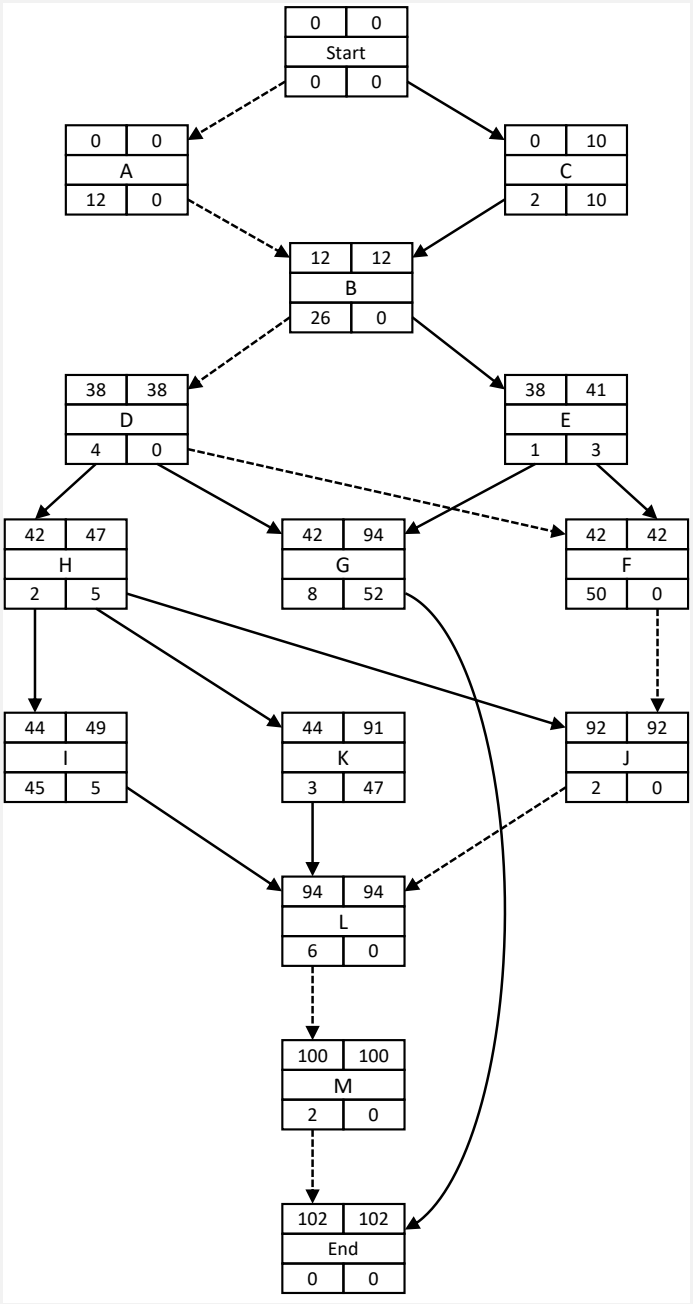
Examining the network you will see that a number of activities have zero float. This means that these activities *must* start and finish on time, and if these activities are delayed the overall project time will be increased. Therefore these activities are critical to the contract. They will form an interlinked path through the network called the **critical path**.

Example 41 *Draw an activity on nodes diagram for the project management problem in Example 40. Perform a forward pass and backwards pass on nodes to determine all the E_i, L_i and F_i. Identify the critical path.*

Solution to Example 41 *After a forward pass the diagram looks like:*



Solution to Example 41 (continuing from p. 139) *Then a backward pass the diagram looks like:*



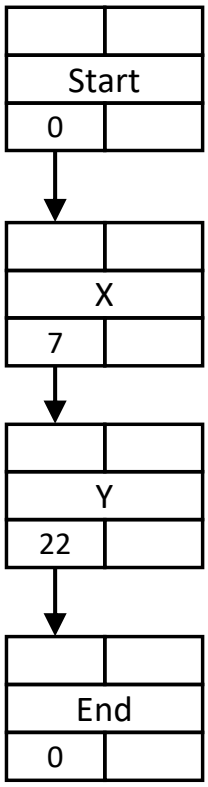
This highlights the critical path as A-B-D-F-J-L-M.

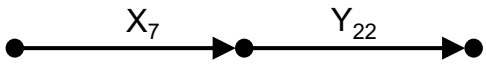
8.3 Activities on Arrows Diagrams

An alternative way of representing activities is an **activity on arrows diagram**. Here activities are arrows labelled by their activity number and duration. Pre-requisites are shown by consecutive arrows. For example consider following project:

Number	Duration	Precedence
X	7	
Y	22	X

Compare its activities on nodes diagram to its activities on arrows diagram:

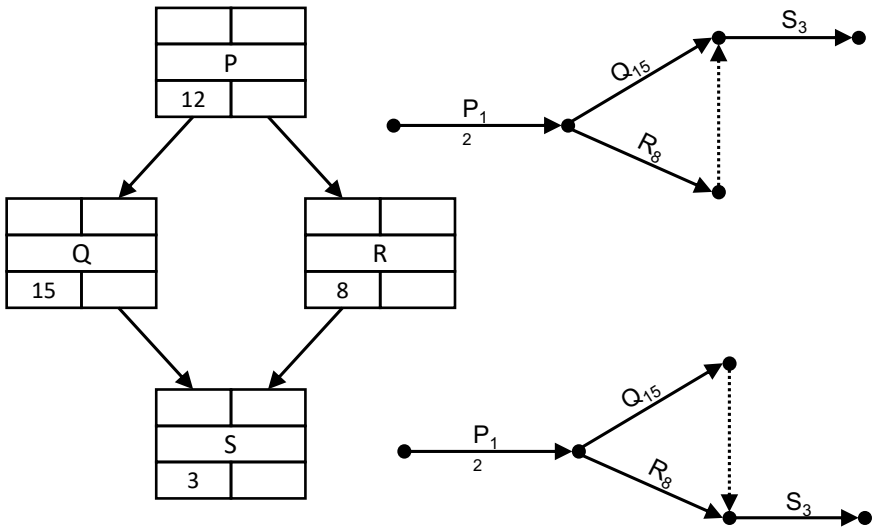




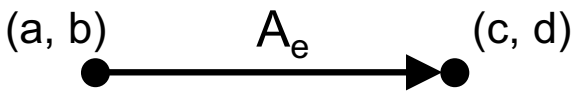
When an activity has more than one pre-requisite, both pre-requisite arrows should meet at the beginning of that activity's arrow. However, one arrow (activity) cannot terminate in two different locations. Similarly, two arrows (activities) cannot begin and end in the same location. Therefore, sometimes dummy activities are required for the diagram to make sense, which have a duration of zero. They are usually drawn with a dashed arrow. For example consider following project:

Number	Duration	Precedence
P	12	
Q	15	P
R	8	P
S	3	Q, R

Compare its activities on nodes diagram to its activities on arrows diagram:



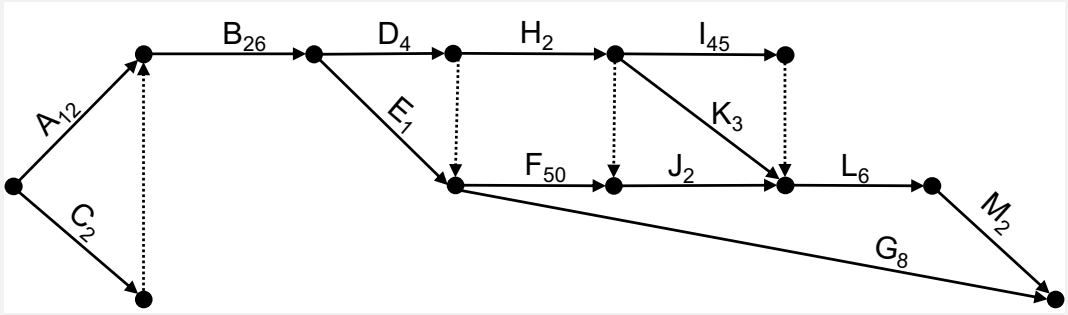
Forward and backward passes can also be performed on these diagrams, with *earliest reaching times* and *latest reaching times* being applied to the nodes connecting the activities now. E.g:



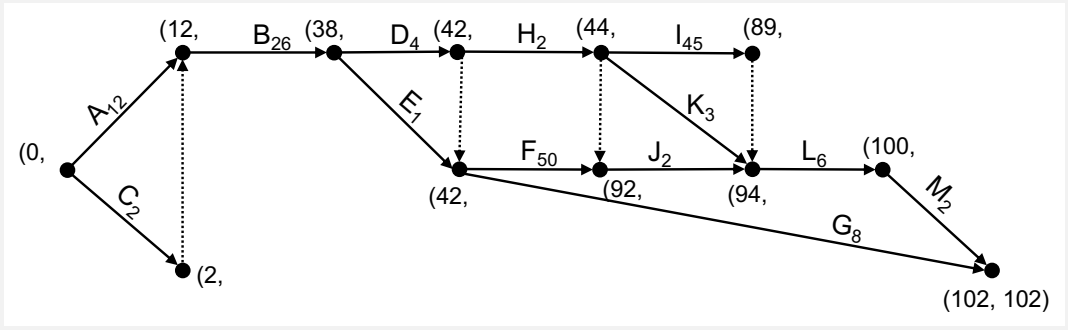
Activity A on the critical path if and only if $a = c$, $b = d$ and $c - a = e$.

Example 42 *Draw an activity on arrows diagram for the project management problem in Example 40. Perform a forward pass and backwards pass on nodes to determine all the E_i, L_i. Identify the critical path.*

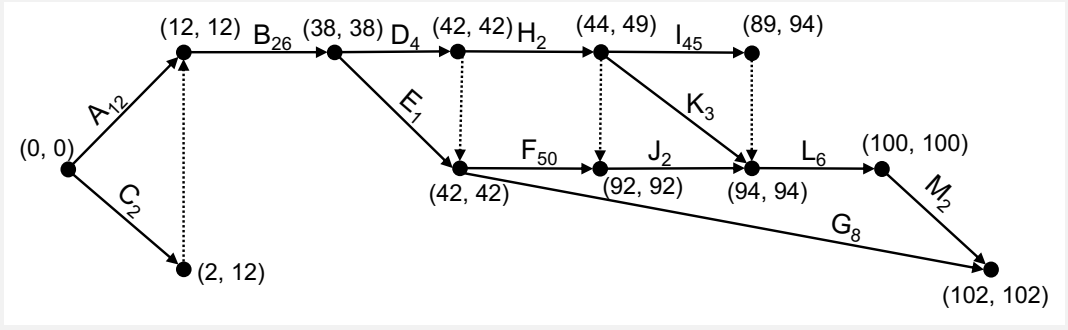
Solution to Example 42 *The initial activities on arrows diagram is:*



After a forward pass the diagram looks like:



Then a backward pass the diagram looks like:



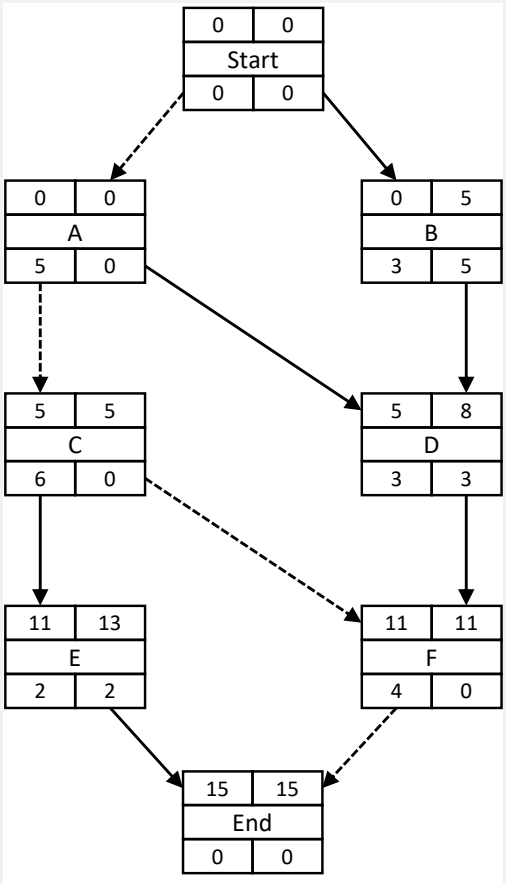
This highlights the critical path as A-B-D-F-J-L-M.

Let's see another example:

Example 43 *Draw both the activity on nodes and activity on arrows diagram for the following project of getting ready in the morning, and find the critical path. What is the absolute quickest that this person can get ready?*

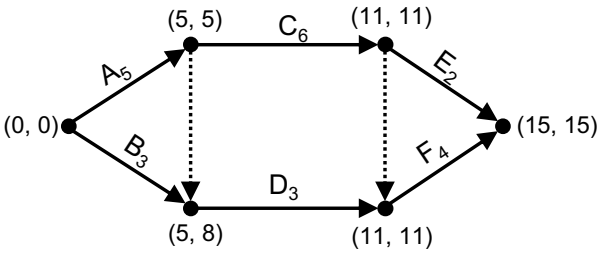
Number	*Activity*	*Duration*	*Precedence*
A	*Shower*	*5*	
B	*Brush teeth*	*3*	
C	*Dry hair*	*6*	*A*
D	*Make up*	*3*	*A, B*
E	*Style hair*	*2*	*C*
F	*Get dressed*	*4*	*C, D*

Solution to Example 43 *The activity on nodes diagram is:*



with critical path A-C-F.

Solution to Example 43 (continuing from p. 144) *The activity on arrows diagrams is:*

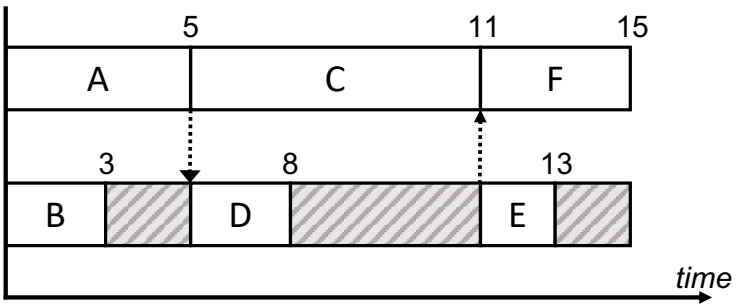


with critical path A-C-F.
The minimum amount of time to complete the project is 15 minutes.

8.4 Gantt Charts

A Gantt chart is another diagrammatic representation of a project. It has an x-axis representing time, and uses rectangles to display the durations of the activities. Dotted arrows can still be used to show pre-requisites. This diagram, when drawn accurately and to scale, can highlight the float times of each non-critical activity.

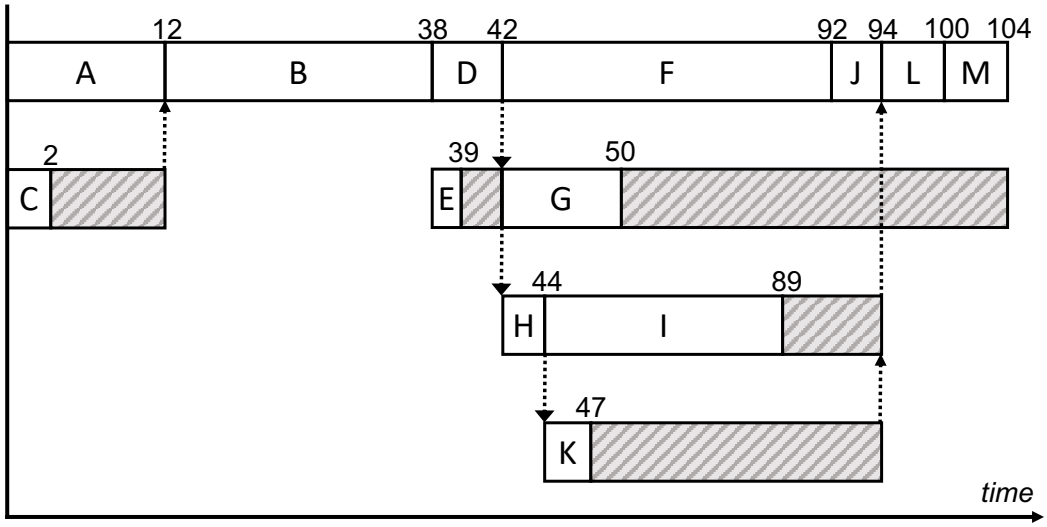
The following Gantt chart is for the project given in Example 43:



Some steps to help:

1. Draw the critical path consecutively first.
2. Their positions will determine the start and end times of the non-critical activities.
3. Gaps in the Gantt chart represent floats.
4. Sometimes start dates of non-critical activities are determined by other non-critical activities, in which case floats will overlap, called a shared float.

For example consider the Gantt chart of the project given in Example 40:



The float between time 89 and 94 is a shared float, shared between activities I and K, as both are pre-requisites to L, along with the critical activity J.

This Gantt chart can help us answer some questions, but also identifies questions needing further analysis.

Example 44 *Using the Gantt chart above, which of the following questions can we answer immediately?*

1. *What happens if the duration of A increases by 2?*
2. *What happens if the duration of F decreases by 5?*
3. *What happens if the duration of C decreases by 1?*
4. *What happens if the duration of C increases by 3?*
5. *What happens if the duration of C increases by 12?*

Solution to Example 44 *We get:*

1. *What happens if the duration of A increases by 2?*
   As A is critical, the duration of the project increases by 2.
2. *What happens if the duration of F decreases by 5?*
   As F is critical we do not know. We might have to determine a new critical path.
3. *What happens if the duration of C decreases by 1?*
   As C is not critical it would have no effect on the duration of the project.
4. *What happens if the duration of C increases by 3?*
   As C has a float of 10, and $3 < 10$, then this would not change the critical path and would have no effect on the duration of the project.
5. *What happens if the duration of C increases by 12?*
   As C has a float of 10, and $12 > 10$, then this would change the critical path, which we would have to determine.

8.5 Crashing

Crashing is the act of reducing the duration of activities at some cost. For example we might hire more builders to build a house more quickly. In such cases we might know two extra pieces of information about each activity:

- **Crash time**: the minimum duration an activity can be achieved in.
- **Cost of reduction**: the extra cost required to achieve the crash time.

Our task is to find the reductions that will lead to the shortest project duration with the least cost.

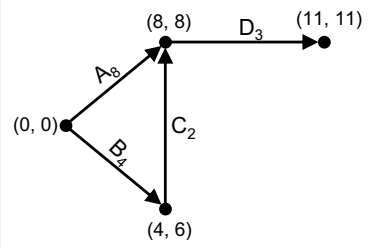
In order to do this we will work iteratively: reducing the activity on the critical path with least cost by 1 each time (unless reducing by more clearly will not change the critical path). Between each iteration we must recalculate the critical path. If there are two parallel critical paths then we may need to consider combinations of reductions.

Example 45 *Consider the following project with given crash times and reduction costs:*

Number	*Duration*	*Precedence*	*Crash Time*	*Cost*
A	*8*		*4*	*£1200*
B	*4*		*2*	*£200*
C	*2*	*B*	*1*	*£400*
D	*3*	*A, C*	*2*	*£300*

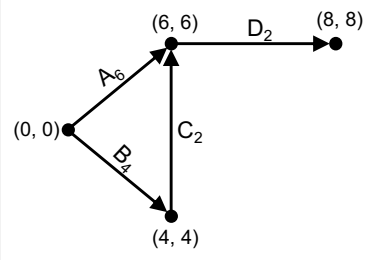
How can we reduce the overall project time by the most, with least cost?

Solution to Example 45 *First note that the unit cost reduction for A is £300, for B is £100, for C is £400, and for D is £300. Draw the initial activity on arrows diagram and find the critical path:*

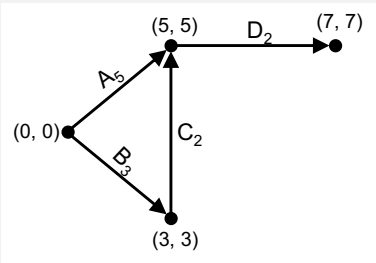


which gives a critical path of A-D. We can reduce D by 1 and A by 2 before the critical path changes.

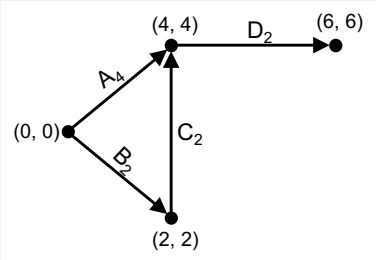
Solution to Example 45 (continuing from p. 147) *Giving:*



we now have two critical paths, A-D and B-C-D. D cannot be reduced any more, and reducing A, B, or C individually will not effect the overall project duration. So we need do consider combinations of reductions. We can either reduce A and B together with a cost of £300+£100=£400, or we can reduce A and C together with a cost of £300+£400=£700. Choose to reduce A and B by 1:



again there are two critical paths, A-D and B-C-D. We have the same options again: reduce A and B together with a cost of £300+£100=£400, or we can reduce A and C together with a cost of £300+£400=£700. Choose to reduce A and B by 1:



and again there are two critical paths. However both options for reduction include reducing A, which cannot be reduced further as we have reached the crash time, so we are done. We should:

- *reduce A by 4 for 4 × £300 = £1200,*
- *reduce B by 2 for 2 × £100 = £200,*
- *reduce D by 1 for £200.*

This means a total reduction in the project time from 11 to 6, for a cost of £1600.

Let's look at one final example:

Example 46 *Consider the following project with given crash times and reduction costs:*

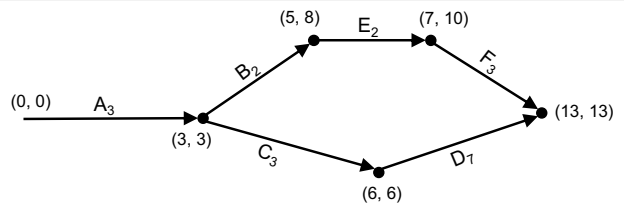
Number	*Duration*	*Precedence*	*Crash Time*	*Crash Cost*
A	*3*		*1*	*£75*
B	*2*	*A*	*1*	*£5*
C	*3*	*A*	*1*	*£30*
D	*7*	*C*	*3*	*£40*
E	*2*	*B*	*1*	*£10*
F	*3*	*E*	*1*	*£25*

There is a budget of £50, what is the minimum time we can crash the project, and how?

Solution to Example 46 *First note the per-unit crash cost for each task:*

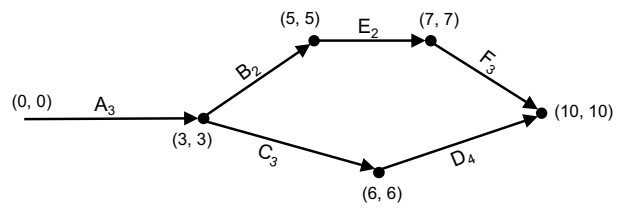
Number	*Duration*	*Precedence*	*Crash Time*	*Crash Cost*	*Per-unit Cost*
A	*3*		*1*	*£75*	*£75*
B	*2*	*A*	*1*	*£5*	*£5*
C	*3*	*A*	*1*	*£30*	*£30*
D	*7*	*C*	*3*	*£40*	*£10*
E	*2*	*B*	*1*	*£10*	*£10*
F	*3*	*E*	*1*	*£25*	*£25*

Now draw an activities on arrows diagram and find the critical path:



which gives a critical path of A-C-D. All can be crashed, and the cheapest activity to crash is D. Looking closely, we can see that we can reduce this by 3 and still not change the critical path, so we crash D by 3, at a cost of £30.
Total running cost = £30.

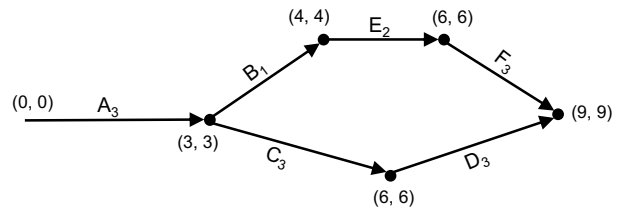
Solution to Example 46 (continuing from p. 149) *Redrawing the activities on arrows diagram:*



which gives two critical paths: A-B-E-F and A-C-D, with the sections B-E-F and C-D parallel to one another. This gives us seven options for crashing:

- *reduce A by 1 for £75*
- *reduce B and C by 1 each for £5 + £30 = £35*
- *reduce B and D by 1 each for £5 + £10 = £15*
- *reduce E and C by 1 each for £10 + £30 = £40*
- *reduce E and D by 1 each for £10 + £10 = £20*
- *reduce F and C by 1 each for £25 + £30 = £55*
- *reduce F and D by 1 each for £25 + £10 = £35*

The cheapest option is to reduce B by 1 and D by 1, costing £15.
Total running cost = £45.



Any crash combinations would now cost more than £5, which is all we have left in the budget. So we stop. We can achieve a project duration of 9 time units, but reducing D by 4 and B by 1, at a total cost of £45.

Further Reading

- "Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling". **William J. Stewart**. ISBN: 978-0-691-14062-9 (2009)
- "Simulation: The Practice of Model Development and Use". **Stewart Robinson**. ISBN: 978-1-137-32802-1 (2014)
- "Operations Research: Applications and Algorithms". **Wayne L. Winston & Jeffrey B. Goldberg**. ISBN: 978-0-534-38058-8
- "Applied Mathematics with Open-Source Software: Operational Research Problems with Python and R". **Vincent Knight & Geraint Palmer**. ISBN: 978-0-367-33998-2 (2022)
- "Ciw's Documentation": https://ciw.readthedocs.io/