MANCHESTER METROPOLITAN UNIVERSITY

Artificial Intelligence in Train Scheduling Problems

by

Kieran Molloy

A dissertation submitted in partial fulfilment for the degree
BSc. (Hons) Mathematics

in the
Science and Engineering
Department of Computing and Mathematics

May 2020

Plagiarism Declaration

With the exception of any statement to the contrary, all the material presented in this report is the result of my own efforts. In addition, no parts of this report are copied from other sources. I understand that any evidence of plagiarism and/or the use of unacknowledged third party materials will be dealt with as a serious matter.

Signed:

_____

Date:

_____

MANCHESTER METROPOLITAN UNIVERSITY

Abstract

Science and Engineering

Department of Computing and Mathematics

BSc. (Hons) Mathematics

by Kieran Molloy

Investigating Genetic Algorithm and Machine Learning applications to the Train Scheduling Problem (TSP). By creating a live data connection to Network Rail and analysing the data, suggestions for junction/track layouts can be made. Modelling the TSP as a Integer Linear Programming (ILP) Problem then using that to construct a Genetic Algorithm framework for further work. Using Swiss Rails crowdAI framework, a discrete event simulation program is created in python, using delayed q-learning to optimise decision making for collisions, for pre-defined instances.

Acknowledgements

I would first like to thank my project advisor Dr Philip Sinclair of the School of Computing and Mathematics at Manchester Metropolitan University. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to acknowledge Nick Watson of Grand Central, and Craig Leaper of East Midland Railway for allowing me to come and ask questions about their operations.

Finally, I must express my very profound gratitude to my parents, especially to my mother for proof reading, and to Niamh for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this project. This accomplishment would not have been possible without them. Thank you.

Kieran Molloy

Contents

List of Figures

List of Tables

Listings

Abbreviations

TSP	**T**rain **S**cheduling **P**roblem
TRSP	**T**train **R**e-**S**cheduling **P**roblem
AI	**A**rtificial **I**ntelligence
ML	**M**achine **L**earning
RL	**R**einforcement **L**earning
GA	**G**enetic **A**lgorithms
EA	**E**volutionary **A**lgorithms
JSON	**J**ava**S**cript **O**bject **N**otation
CIF	**C**ommon **I**nterface **F**ile
XML	e**X**tensible **M**arkup **L**anguage
REST	**RE**presentational **S**tate **T**ransfer
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
API	**A**pplication **P**rogramming **I**nterface
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
CORS	**C**ross **O**rigin **R**esource **S**haring
CRS	**C**omputer **R**eservation **S**ystem
TOC	**T**rain **O**perating **C**ompany
NLC	**N**ational **L**ocation **C**odes
TIPLOC	**TI**ming **P**oint **LOC**ation
STANME	**STA**tion **N**umber Names
STANOX	**STA**tion **N**umbers

Chapter 1

Introduction

This study is based on the Artificial Intelligence methods applied to the Train Scheduling Problem. Network Rail hosts one of the busiest railway systems in Europe, often the infrastructure is situated in places where expansion is not possible and therefore it is important to be able to get as much 'value' as possible from the existing infrastructure. Network Rail is the owner of all railway assets in the UK, however day to day running is allocated to Train Operating Companies (TOC's) who are businesses looking to make a profit; for TOC's the 'value' lies in being a profitable business which entails selling tickets for services and not having service performance fines. Where as for Network Rail the 'value' is in providing robust infrastructure and safety of the users of the railway. The planning process for all rail operations can be summarised in Figure 1.1, the key stages are Strategic, Tactical and Operational. The strategic level is the highest level of the hierarchy and consists of Infrastructure and Demand Analysis. The middle tier is Tactical Planning, this consists of Line Planning, Train Timetabling, Unit Assignment and Crew Scheduling. This stage can be thought of as the stage where optimisation can have the largest impact. Line Planning will be performed solely by Network Rail, and is in the Rail Franchise Agreement each TOC possesses. Each Line will have a preset requirement of services outlined in the Franchise Agreement, TOC's have freedom for at which time they operate but little freedom for calling points, origin or destination. After each individual TOC has calculated their preferential

times, this information is given to Network Rail to calculate the final timetable. Each TOC will have their own processes for rolling stock and crew scheduling, and have freedom to allocate as they wish.

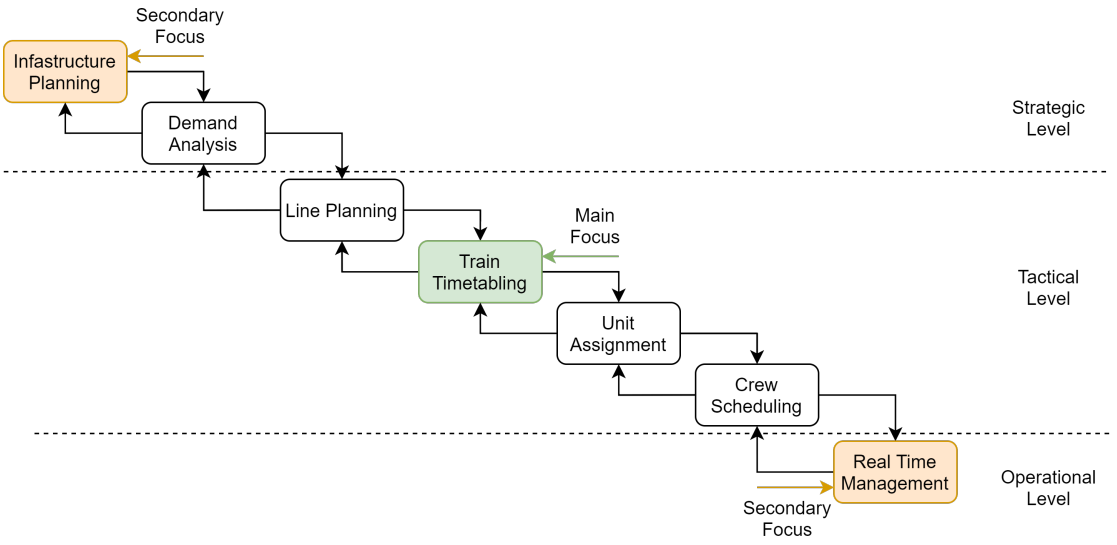


FIGURE 1.1: Hierarchical Planning Process for Urban Rail

1.1 Research Questions

The various different methods of Artificial Intelligence and techniques of applying them provide space for further research. To structure this research and choose the direction to take it, several research questions are given in this section and are revisited in the conclusion (Chapter 9). There are three questions for each subject area, the areas are structured as follows; Chapter 3, Chapters 5, 6, 7 and Chapter 8.

1. Chapter 3

   (a) *What benefits can come from observing all trains?*

   Trains often become delayed due to other trains out of their control; such as junction blocking or different stopping patterns. These can cause massive knock on delays where the full effect can only be seen from a pure data perspective. Additionally

(b) *Can monitoring trains give an advantage for identifying areas prone to delay?* Monitoring the network can give in depth data for all sections, in a mass of all the data collected, anomalous data will be easily identifiable. Additionally, time is critical to establishing high utilisation times of the day; such as AM/PM rush hour.

2. Chapters 5, 6, 7

   (a) *What are the biggest obstacles for Artificial Intelligence in Railway Scheduling?* Investigate what problems AI needs to overcome to be used for solving optimisation problems in the industry

   (b) *Can Reinforcement Learning be applied to the Train Scheduling Problem?* The core focus of this paper is work into Reinforcement Learning in Chapter 6, this question could also be phrased as Should Reinforcement Learning be applied to the Train Scheduling Problem.

3. Chapter 8

   (a) *Why are trains often delayed on the Manchester Corridor?* Investigate the potential reasons for Manchester being a delay hotspot

   (b) *What can be done to alleviate pressure on junctions in Manchester?* Suggesting potential improvements to increase Trains Per Hour (tph) (and therefore decrease utilisation rates) whether it be infrastructure, scheduling or other types of improvements.

To summarise the research questions, the aims are: (1) to setup a way of storing all live train data (2) explore methods of applying Artificial Intelligence to the TSP and (3) critically evaluate Manchester Junctions and give suggestions for improvement

1.2 Research Scope

This research will contain three main areas. The first part is data engineering, that consisting of collecting the data, formatting it in a way that can later be referenced

and setting up a framework to easily reference it from any number of devices. The second part investigates potential scheduling strategies to solve the Train Scheduling Problem. The final part is a review of Manchester's infrastructure and suggestions on improvement. The most important part is investigating scheduling strategies, as it is the most difficult problem in addition the most dominant factor in the experiments.

1.3 Research Relevance

Optimisation Problems are appealing to business of all kinds due to it either minimising costs or maximising money. For non-profit organisations, it is appealing due to maximising 'value'. Machine Learning represents an area of 'un-tapped' potential for optimisation, this research will contribute to the evolving landscape of literature in the area of Machine Learning in Railway Scheduling. Although this research will focus on Railway, the theory behind the application can be applied to other transportation methods such as Airlines, or Shipping. The approach could also be extended to be applicable to logistic environments, or multiple forms of transport.

1.4 Document Outline

The brief document outline is as follows:

1. Chapter 1 - Introduction
2. Chapter 2 - Literature Review
3. Chapter 3 - Collecting Network Data
4. Chapter 4 - AI Pre-requisite Concepts
5. Chapter 5 - Genetic Algorithm TSP
6. Chapter 6 - Reinforcement Learning TSP
7. Chapter 7 - MILP TRSP

8. Chapter 8 - Manchester Corridor Analysis

9. Chapter 9 - Conclusions

Chapter 2

Literature Review

This chapter provides a review of to-date research on the Train Scheduling Problem, with a focus on the area of Machine Learning and how Machine Learning techniques have been applied to the TSP. A secondary focus will be on presenting the advantages, disadvantages and knowledge gaps in a ML TSP. The chapter is organised into three sections. The first section introduces the Train Scheduling Problem and the different methods of solving it. The second second discusses Artificial Intelligence in a broad sense, then focusing on Genetic Algorithms as well as Machine Learning, its applications to the TSP and general scheduling problems. Especially looking at Reinforcement Learning, the current scene of Reinforcement Learning Algorithms and applying them to scheduling problems, culminating with Reinforcement Learning in the TSP. The final section will briefly discuss scheduling problems for other vehicles such as buses or aeroplanes.

2.1 Rail Data Feeds

Quite extensive hobbyist work has been performed on the Network Rail Data streams, a large chunk of which is documented on the wiki (*Open Rail Data Wiki* 2020). In addition to this the only previous academic study in this area (Lipinski 2015) however this focuses on a mobile phone app and is built on a very different architecture, but the principals of the systems are similar. Other popular hobbyist

systems are Traksy, OpenTrainTimes and Raildar. Some inspiration is taken from these projects, but there is no open source information for how any aspect of their systems run.

2.2 Train-Focused Problems

Railway Scheduling has been studied from every angle in literature, due to both the economic significance of delays and its mathematical complexity. Traditional optimisation can be divided into 3 distinct categories;

1. Analytical Solution
2. Mixed Integer / Quadratic Programming
3. Heuristics

2.2.1 Analytical Solution

Studies that use Analytical techniques to solve railway scheduling are able to model broad characteristics such as congestion (Petersen 1974) and delay propagation (Castillo et al. 2008); however studies such as these make assumptions about random processes or about periodicity of timetables. The periodicity is good for application in the UK, but are not useful for truly solving practical scheduling instances where demand is not periodical.

2.2.2 Optimisation Programming

The second category is Integer Linear Programming or Quadratic Programming. Exact solutions to such formulations are limited in terms of scale and complexity, successful models are usually based on a small set of arrival and departure events. (Gestrelius et al. 2017),(Fischetti and Monaci 2015).

2.2.3 Heuristics

Heuristic methods are probably the most powerful of the three methods, and also the most popular. (Higgins et al. 1997),(Furini and Kidd 2013)

2.3 Artificial Intelligence

Artificial Intelligence provides an area of improvement for all types of problems, which is especially true for scheduling problems where time windows can be up to 6 months of planning. Tormos et al. 2008 developed a genetic algorithm implementation for railway scheduling problems that were used to solve real world instances with good performance.

2.3.1 Machine Learning

Machine Learning represents the most likely area of AI to be successful for Scheduling Problems, whereby it is able to overcome the heuristic solvers currently in place, to date Machine Learning has not been widely used for Railway Scheduling partly due to unavailability of the learning data required.

2.3.2 Machine Learning in Railway Scheduling

Swiss Federal Railway published their plan for overcoming data sparseness by running in depth simulations in Nygren et al. 2017 this application is in Dispatching, and mentions that big data does not mean big information. In Jean-Francois Cordeau 1998, an overview of railway optimisation models are shown, the authors make a point that air transportation optimisation developed far faster than railway due to railway optimisation being far larger and difficult. Since then, computation ability increased which led to applications able to the Heuristic solvers mentioned above. It was not until the development of sensor technologies such as 5V (Volume, Velocity, Variety, Value and Veracity) until Machine Learning was more viable for applications in rail, however not for the TSP (Thaduri et al. 2015). Supervised learning has been widely used for mimicking human controllers, but is limited to conflict resolution (Dündar and Şahin 2013). Additionally, there are a variety of

studies based on speed profiling of train units that use Reinforcement Learning (Lu et al. 2013). Other learning approaches in the railway domain tend to focus on shunting trains within yards (Hirashima 2011)(Hirashima 2012) or subway models (Yin et al. 2014). Recent surveys of algorithms for railway scheduling (Turner et al. 2016), cover exact approaches, simulation models, constraint propagation, alternative graphs, heuristics and expert systems, but no previous Reinforcement Learning methods. The only previous Reinforcement Learning application that can be found (Khadilkar 2019)

2.3.3 Machine Learning in Planning & Scheduling

Due to the lack of Reinforcement Learning in Railway Scheduling, one has to rely on solutions to the job shop scheduling (Zhang and Dietterich 1995), but these too, are few in number and are used as a tool to fix infeasibility as well as (Littman 1994) to fix markov decision process logic.

2.3.4 Reinforcement Learning

A recent study (Khadilkar 2019) appears be one the only other reinforcement learning for railway scheduling. It uses a q-learning approach similar to the one included in this research, however there are key distinctions between the approaches. The size of the state space is a key point of research in the prior study, and a focus on training and memory management is made. The specificity of the model is not an issue, as the key result is testing the different state spaces for all problem instances. The study does have the same aim as this research, which is developing schedules from any initial state. Several studies have used reinforcement learning in broadly related areas, a survey of those summarise important studies (L. P. Kaelbling et al. 1996). The same author earlier produced key work in RL methods (L. Kaelbling 1994). An airline seat allocation algorithm (Gosavi et al. 2002) is useful for the large-scale optimisation using reinforcement learning methods

Chapter 3

Rail Data Feeds

As we enter the information age, we are inundated with various types and formats of data that must be interpreted in a positive manner. There is tonnes of data readily available to the average person provided by Network Rail and the UK Government, and it is important to use this data when making informed decisions on and about the network such as timetabling and routing. The masses of data can be used to find artificial choke points that may not obviously be a problem region, or problem trains that may not cause themselves be delayed, but cause other trains to become delayed, for example, a Manchester Airport to Middlesbrough train may not itself be delayed, but due to the track it requires at Ordsall Junction being blocked, means that 5 tracks are blocked at once (Further analysis in Chapter 8). Without data, it may not be possible to realise this.

3.1 Introduction

This chapter documents the process of setting up a service that monitors all trains in the UK and stores them in a database, additionally setting up a service to query the database to receive information. The first section defines the data sources and briefly mentions the data streams (the specific 'types'). The second section describes the design of the system from a server-oriented architecture and briefly mentions the methods of collecting and storing the data. The third section goes

though the system implementation and technologies used to achieve this, additionally the database designs used, and the REST Service to query the database.

3.2 Sources

This chapter relies on quality data sources, with consistent formats that can easily be interpreted automatically and filtered automatically. Equally a high level of detail is required for them to be of any use. This project utilises two main sources of data.

3.2.1 Network Rail

Network Rail provide a number of operational data feeds available to developers. They state on their website (*Network Rail Data Feeds* 2020) that by providing access to these feeds, they hope to encourage the development of new products of interest to those who use the railway.
The available feeds are:

- SCHEDULE
- MOVEMENT
- TD
- TSR (Temporary Speed Restrictions)
- VSTP (Very Short Term Plan)
- RTPPM (Real-Time Public Performance Measure)
- SMART
- Corpus
- BPLAN
- Train Planning Network Model

This paper will utilise most of these, and will be described later. More information on these data streams and how they are used is in the additional file

3.2.2 National Rail (Darwin)

Darwin is the primary train tracking system in the UK, and is used in all stations for their Departure Boards. It uses many data sources including Network Rail's to provide real-time arrival and departure predictions. The information from Darwin is far more detailed than that from Network Rail, however is significantly over-detailed and complicated for the scope of this project. The Historic Service Performance will be used to provide additional reliability metrics that were not collected with the system created for this paper (due to the HSP giving a time period of up to one year). The data streams provided by the Darwin system are:

- LDB Webservice (PV)
- LDB Webservice (Staff Version)
- Darwin Timetable
- Darwin Push Port
- Historic Service Performance (HSP)

More information on these, and how they are used are also in the additional file.

3.3 System Design

The systems designed for this project follow a service-oriented architecture, and all operate independent of each other. This allows for greater flexibility when testing or creating new features. There are 4 separate systems that integrate with each other using REST

1. Streamer
2. REST Service
3. Website

4. Android App

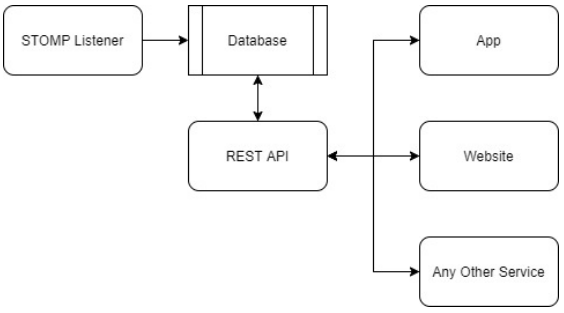


FIGURE 3.1: Server-Oriented Architecture

The service-oriented architecture allows for massive scalability as each system can be replaced individually as long as they produce the same output.

3.3.1 Finding the Data-stream

Designing the Streamer (or more accurately the listener) was based on the snippet provided by the Open Rail Wiki (*Open Rail Data Wiki* 2020) and is shown in Listing. 3.1

```
var prettyjson = require('prettyjson'),
    StompClient = require('stomp-client').StompClient;

var destination = '/topic/TRAIN_MVT_ALL_TOC',
    client = new StompClient('datafeeds.networkrail.co.uk', 61618, '
   your-email', 'your-password', '1.0');

client.connect(function(sessionId) {
    console.log('Trying to connect...');
    client.subscribe(destination, function(body, headers) {
        console.log(prettyjson.render(JSON.parse(body)));
    });
});
```

LISTING 3.1: Node.JS Example Stomp Listener

Listing.3.1 shows a simple STOMP application in JavaScript that only listens for all TOC changes, and then prints them as they come. Using a program similar to this allows us to confirm the data format shown on Open Rail Wiki is correct

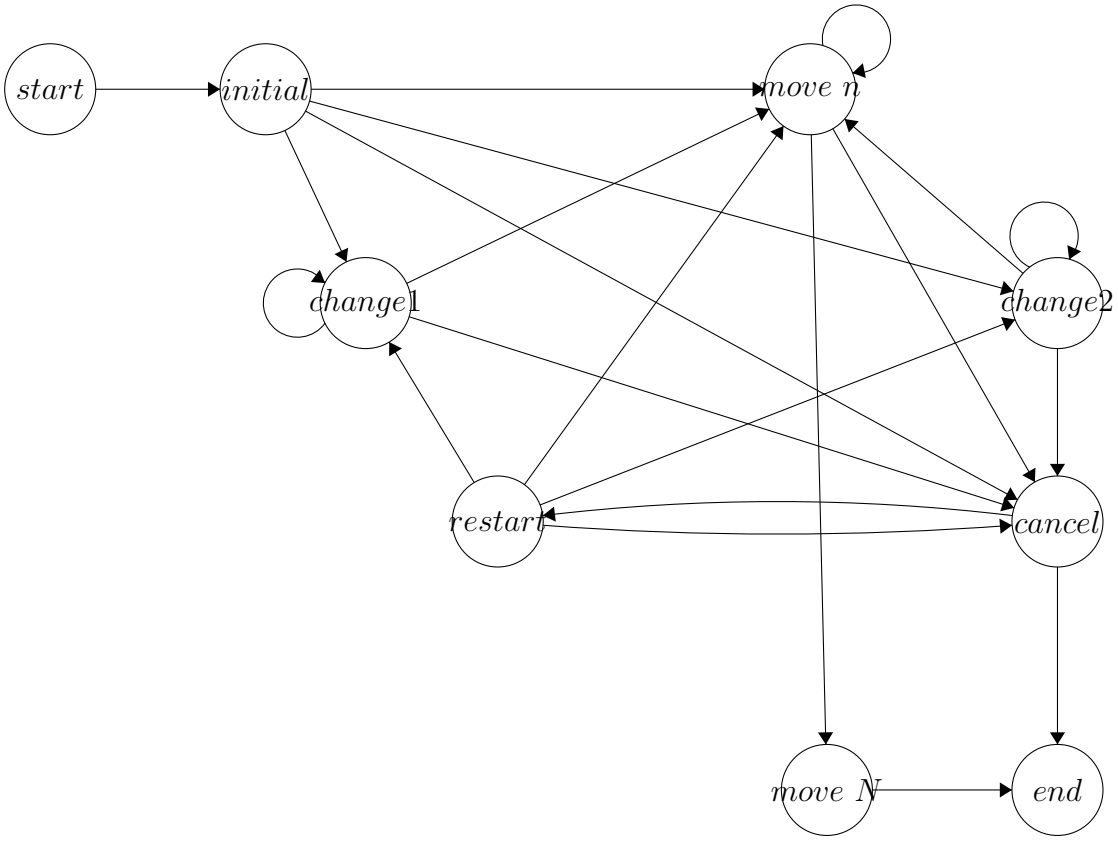


FIGURE 3.2: Network Rail TRUST Finite State Machine

and, as such, the databases that will store the data must be designed, as simply printing the data is not the final solution. The system requires some static information, which are 'reference','smart' and 'corpus'. These are direct imports and are described in more detail in the additional file.

The 'schedule' collection is initialised with a full import, but then uses a daily import to keep up to date.

The 'trains' collection records every train that's been seen on the network. This includes trains which have been TRUST activated, have been seen by a Train Describer. The records are a linking of the SCHEDULE, TD and Train Movement feeds, as well as the SMART/CORPUS data about their current location. A train implementation can be envisaged as a finite state machine, see Figure 3.2

3.3.2 CIF

The Common Interface File (CIF) format is the industry standard for transfer of schedules from Network Rail ITPS to other downstream information systems. Most feeds supply both JSON and CIF, however CIF is usally more detailed however it is far more complex and harder to interpret/parse. Although JSON is used in other areas of the project, CIF is used for this section due to it being the standard format in this area, as well as being more detailed.

3.4 System Implementation

The system implementation is broken down into two sections; tools used and descriptions. Where tools used briefly discusses the technologies that are used to achieve the implementation. And the descriptions discuss each seperate system and the tools behind it.

3.4.1 Tools Used

All tools used in development of NRDF Service are described here, since it is a complex problem with various purposes and requirements.

3.4.1.1 Javascript

Javascript forms the entire basis of the Streamer, REST Service and Website. It was chosen because it was able to perform all 3 tasks well. If more data needed to be processed (if Darwin was to be used) then Javascript could not keep up, and a solution would need to be made using a more strongly typed program in C or a memory-optimsed Python program

3.4.1.2 STOMP

Network Rail API uses STOMP to transmit all data, STOMP (Simple (or Streaming) Text Orientated Messaging Protocol) provides an interoperable wire format so that STOMP clients can communicate with any STOMP message broker to

provide easy and widespread messaging interoperability among many languages, platforms and brokers. (see *STOMP* 2020)

3.4.1.3 Node.js

Node.JS is a JavaScript runtime environment, all JavaScript in this paper uses Node.JS. Additionally, due to the limited time-scope of this project it was infeasible to create a project devoid of dependencies, so Node.js manages the required dependencies including *mongoStream* (MongoDB interface), *bunyan* (JSON Logger) and also *stompit* (STOMP interface).

3.4.1.4 CIF

CIF is one of the two data file types used in this paper, and is described in more detail in Section. 3.3.2. CIF is used for all live data, including parsing the TRUST and TD data streams.

3.4.1.5 JSON

JSON (JavaScript Object Notation) is a data interchange text format (*JSON* 2020) and is the second data file type used in this paper. It is used for static data, such as the streamer config file, and logging. It is easy to read and write and is language independent. Although Network Rail also uses JSON for sending out their movement messages, it is not the preferred choice. It is also used in the REST Service when returning the data after receiving a request.

3.4.1.6 MongoDB

MongoDB was chosen for its versatility as a NoSQL table, and is the core database for the project. The streamer saves data to the table, and the REST Service queries the database.

3.4.1.7 Google Maps API

Google Maps is the leading map provider, and has fantastic JavaScript integration as well as lots of documentation.

3.4.1.8 Postman

Postman was used to test all REST Service endpoints, and was key for development of the REST system.

3.4.2 Listener

The Streamer uses Node.JS and Javascript with a STOMP Listener to collect data from Network Rail and inputs that data into a MongoDB database.

3.4.3 Core Database

The core database has multiple collections that all have their purpose briefly described here. For a more detailed description look at the additional document.

3.4.3.1 Association

Stores schedule information relating to what day(s) a specific schedule entry (route) will run.

3.4.3.2 Berths

Stores all train headcodes that have passed through that specific berth for a given berth.

3.4.3.3 Corpus

Stores all entries from the CORPUS system (Codes for Operations, Retail & Planning – a Unified Solution). This can be used to translate STANOX, TIPLOC, NLC, UIC and 3-alpha (CRS) codes to location latitude and longitude.

3.4.3.4 Reference

Stores schedule entry (route) information such as Operator and Timetable Start/End data.

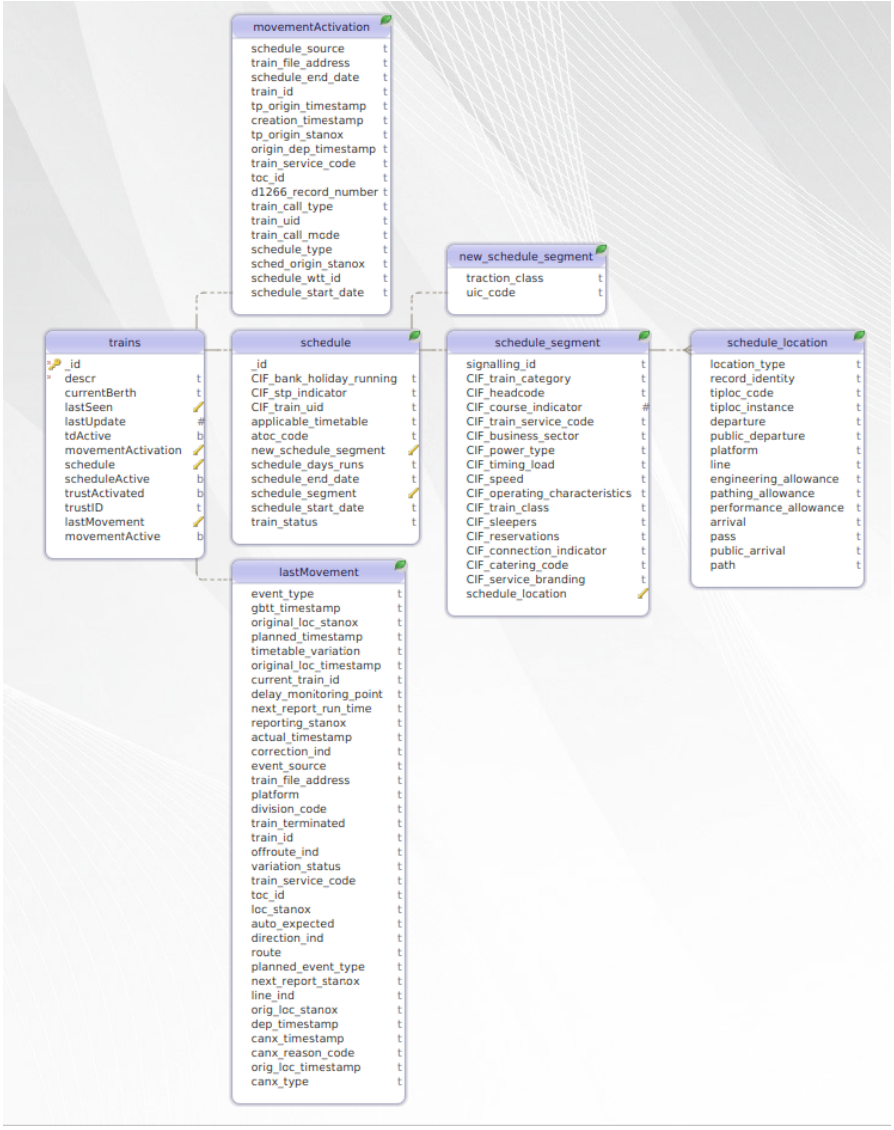


FIGURE 3.3: Relation Diagram of 'trains' db

3.4.3.5 Schedules

Stores every timetable entry, and all related information. It indicates all stations it will call at, and their respective times as well as various operational data.

3.4.3.6 Signals

Stores the current state of each signal (not all signals are in the system, usually only signals in junctions are available).

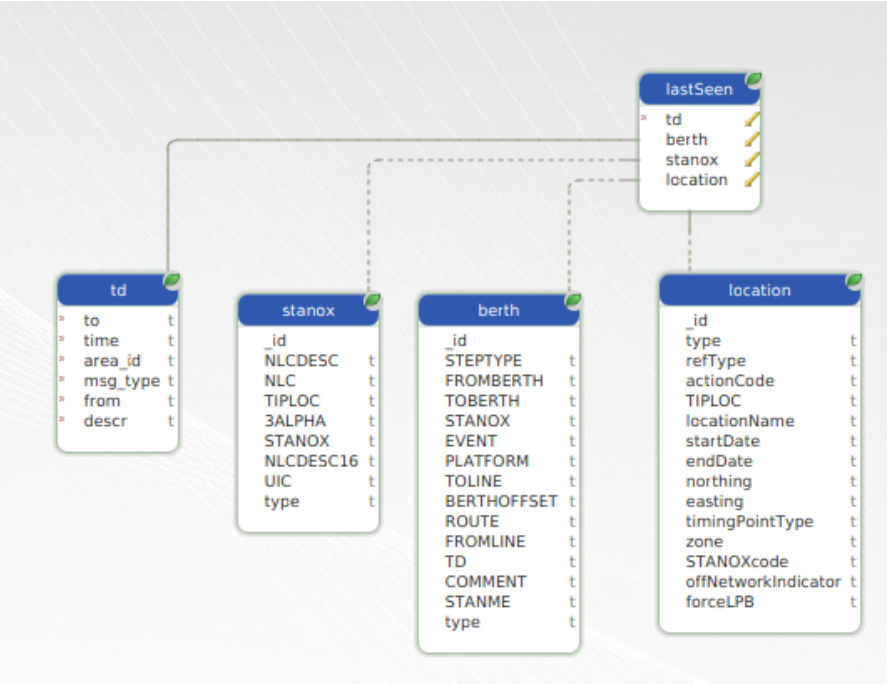


FIGURE 3.4: Relation Diagram of 'lastSeen' db entry

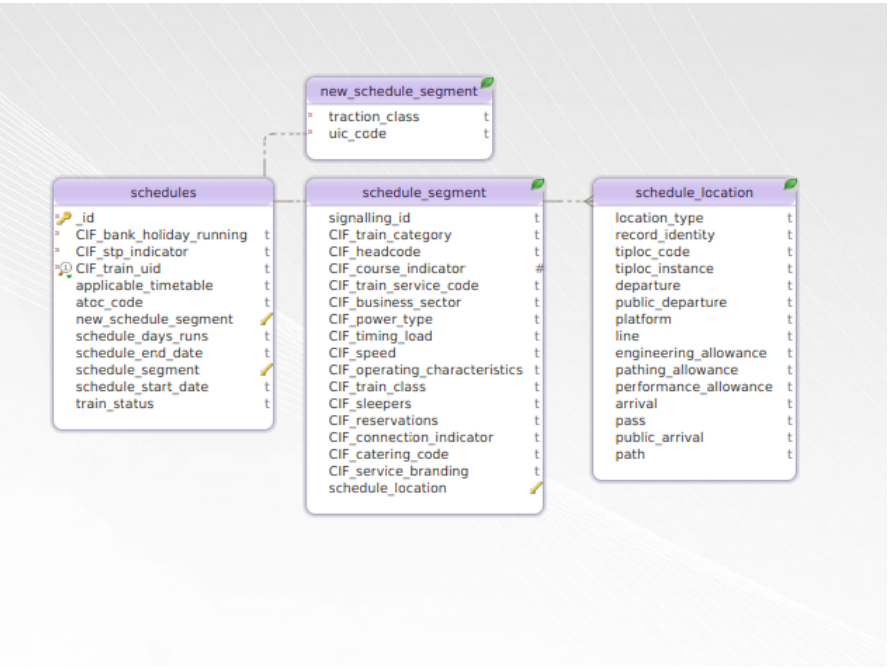


FIGURE 3.5: Relation Diagram of 'schedules'

3.4.3.7 Smart

Stores berth to berth information, to understand how all berths link together.

3.4.3.8 Tiplocs

Stores all TIPLOC (Timing Point Location) entries.

3.4.3.9 Trains

Trains is the key database that is used for the REST API for live information. Every TD Activated Train gets an entry, and trains can also be Schedule Activated, Movement Activated and Trust Activated. A train entry can store all required information for any operation.

3.4.4 Auxillary Database

3.4.4.1 Tocs

Stores all Train Operator's key information.

3.4.4.2 Routes

Stores some routes (this is a database being populated manually, and as such is no where near complete).

3.4.5 REST Service

The REST Service is the basis for all other interactions with the data, and conforms with standard RESTful practices. The REST Service is a Node.JS and Express application to deliver fast and responsive requests. If the service was to become public asynchronous implementation would be a priority, but currently the system works as desired. Requests are made via HTTP.

Example Requests:

```
http://~REST-API~/live/schedules/nt
http://~REST-API~/live/service/W12345
http://~REST-API~/live/all/mco/to/wgn
```

The first request will return the next 25 trains that NT (Northern Rail) will run. The second request will return detailed information about the train with UID

W12345 that exists on the day. The third request will return the next 10 trains that arrive/depart at MCO (Manchester Oxford Road) to WGN (Wigan North Western).

The REST application is technically a plug-and-play design, with a simple change of database string could work for any database with a similar design with minimal overall changes. However for more complex queries, it will need a larger overhaul of code.

3.4.6 Huxley

Huxley is a CORS enabled JSON proxy for Darwin, it was created by James Singleton for cross platform availability. The decision to use this premade service

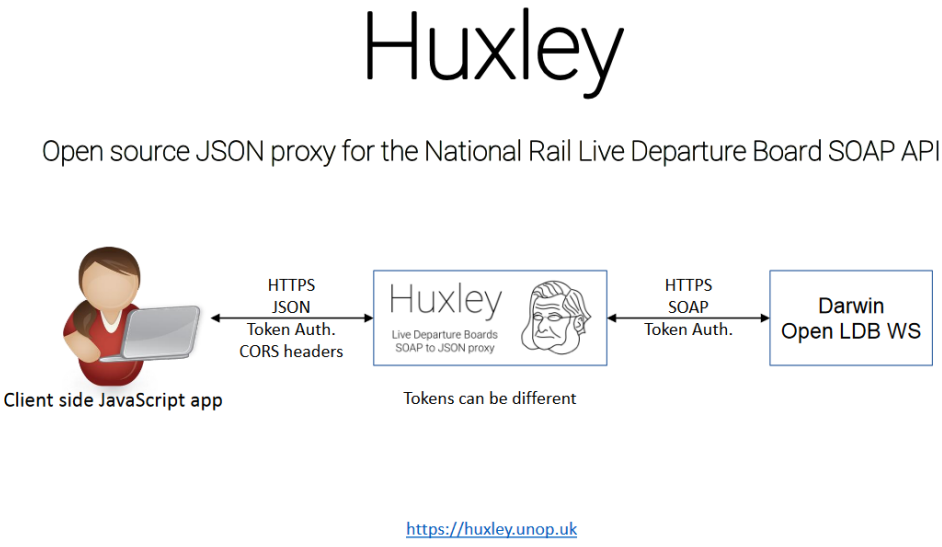


FIGURE 3.6: Huxley System Flowchart (from Singleton 2015)

is due to Darwin being significantly harder to construct a service directly to. Darwin uses SOAP (which requires POST-ing specific XML requests), instead Huxley serves as a middle man (see Figure 3.6) allowing to use GET requests and recieve JSON with the addition of CORS headers. Additionally, Huxley has CRS lookup API integrated, this is not currently implemented however it is a planned improvement. Huxley is used for all non-statistical data, that being any requests for live trains/ live stations is performed by Huxley. Additionally the delay API

is implemented however is used for any purpose other than to check the correct functionality of the core database.

3.4.7 Website

The Website utilises the REST Service, as well as Huxley, to allow a view into the data. Using Node.JS and Express (as well as alot of other dependencies) as a basis for the website backend. It uses an MVC Project Structure, and uses a REST-Style template engine, Pug.JS, to deliver web pages.

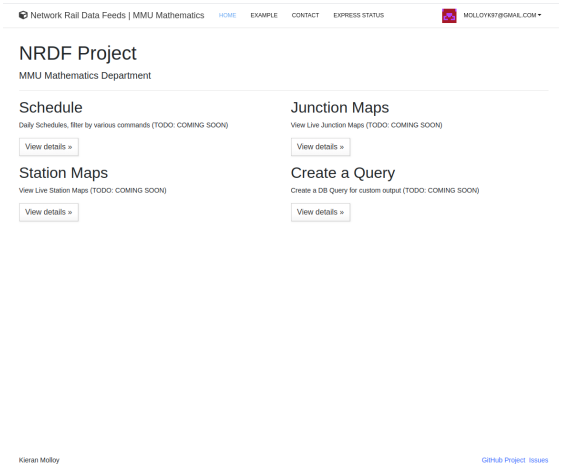


FIGURE 3.7: Homepage of NRDF Website

3.4.7.1 User Functionality

The website has user login functionality including session storage and remembering previous searches, user security used includes password encryption and salting. There are two databases required for functionality, 'sessions' and 'users'.

3.5 Results

This project has delivered 3 key programs, and 2 side programs. The key program is the REST API which powers all services, but this would not be fully functional without the Streamer and Databases as designed here. The website and Android App are simple interfaces for easily accessing the API. The Android app can be

run on any Android device running Oreo 8.0 or higher. However due to limitations with the database and API, must also be connected to MMU Eduroam.

3.6 Discussion

This project has generally been successful, and has delivered everything that was set out. Further work would include extending the API to improve functionality, potentially include more security features to it or tokenisation. The database design could be improved by using referencing instead of what currently happens where data is copied into new documents. As for website and Android App, they could be improved in every way, simple improvements could include UI updates. It would be interesting to include user geolocation to provide a list of nearby stations.

Chapter 4

Artificial Intelligence Concepts

4.1 Introduction

This section introduces key concepts for underlying theories in later chapters (specifically, Genetic Algorithms for Chapter 5 and Reinforcement Learning for Chapter 6). Artificial Intelligence will be key for the Transport Industry, it has the potential to revolutionise all forms of optimisation problems, of which the railway industry is entirely; whether it be minimising timetabling delay, minimising cost or maximising passenger throughput. Current methods for optimisation of a problem as large as this are relatively brute force and take extremely long to solve, a Machine Learning environment would work extremely well for scheduling a train network due to the reinforcement nature of the algorithm. Artificial Intelligence (AI) does not refer to a single technology or concept, but is a set of approaches, methods and concepts which vary in their 'intelligent' trait. EPRS grouped the AI methods into three distinct categories (Reillon 2018); symbolic AI, data-driven AI and future-AI. Symbolic AI includes systems where a human creates a succession of logical rules, transcribed in algorithms, which machines can follow to decide how to act in a given situation. Data Driven AI is an AI that combines machine learning techniques with technologies used for searching and analysing large quantities of data. Future AI include various developments where AI could display

an ever wider range of human capacities, or where AI outperforms humans. This report is only interested in symbolic and data-driven.

4.2 Reinforcement Learning Concepts

Reinforcement Learning requires the definition of a Markov Decision Process, a definition of these can be found in Appendix A.1. There are various popular reinforcement learning methods; SARSA, Temporal Difference, Q-Learning.

4.2.1 Bellman Equations

Bellman equations refer to a set of equations that decompose the value function into immediate reward and discounted future reward:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+1} + \gamma^2 R_{t+3} + \dots)] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s], \end{aligned}$$

and for Q-value:

$$\begin{aligned} Q(s,a) &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a\ \pi} Q(S_{t+1}, a)|S_t = s, A_t = a]. \end{aligned}$$

The recursive update property can be further decomposed to state-value and

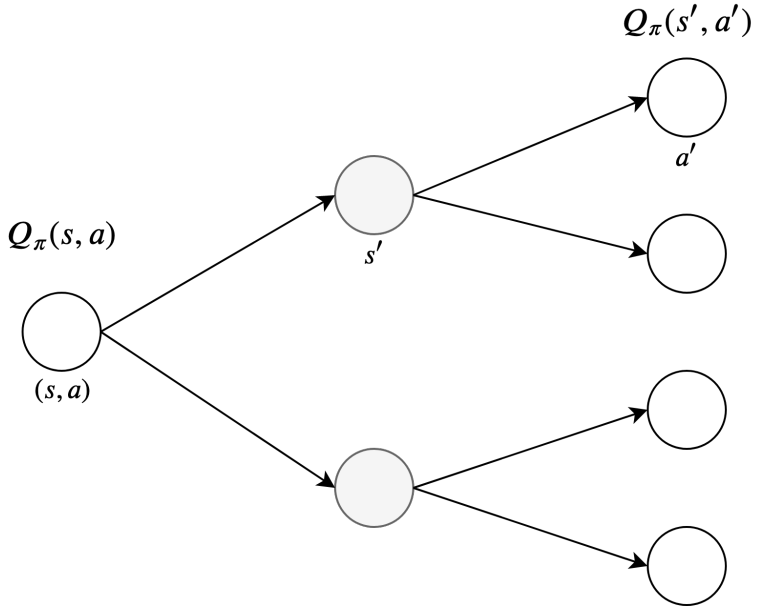


FIGURE 4.1: Bellman Expectation Equation Update state-value and action-value

action-value functions; The policy that is being 'learned/followed' is π

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) Q_\pi(s, a)$$
$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P^a_{ss'} V_\pi(s')$$

Substituting in the new definitions of Q and V respectively,

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) R(s, a) + \gamma \sum_{s' \in S} P^a_{ss'} V_\pi(s')$$
$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a \in A} \pi(a|s) Q_\pi(s, a)$$

If optimal values are the only variable of interest, then the recursive nature can be skipped:

$$V_*(s) = \max_{a \in A} Q_*(s,a)$$
$$Q_*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} V_*(s')$$

Substituting in the new definitions of Q_* and V_* respectively,

$$V_*(s) = \max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} V_*(s')$$
$$Q_*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} \max_{a \in A} Q_*(s,a)$$

If the entire environment is known, the best method of solving is Dynamic Programming however in most situations $P^a_{ss'}$ and $R(s,a)$ are unknown; therefore MDP's cannot be solved by the derivation of Bellman equations and form the basis Reinforcement Learning algorithms.

4.2.2 Dynamic Programming

As mentioned in the preceding section, Dynamic Programming can be used when modelling a wholly known environment to evaluate value functions iteratively by improving policy. A policy evaluation is defined by:

$$\begin{aligned} V_{t+1}(s) &= \mathbb{E}_\pi[r + \gamma V_t(s') | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)(r + \gamma V_k(s')) \end{aligned}$$

for the state value V_π where π is the policy. After evaluating a policy, it is improved upon in a greedy style method

$$\begin{aligned} Q_{\pi'}(s,a) &= \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} P(s', r|s, a)(r + \gamma V_\pi(s')) \end{aligned}$$

where $\pi' > \pi$. This type of method is a Generalised Policy Iteration algorithm, whereby the value function is repeatedly approximated until it approaches the true optimal value by improving the policy. For a policy π, and an improved policy π' which is generated by $\pi'(s) = \max_{a \in A} Q_\pi(s, a)$. The value of π' is guaranteed to be better

$$\begin{aligned} Q_\pi(s, \pi'(s)) &= Q_\pi(s, \max_{a \in A} Q_\pi(s, a)) \\ &= \max_{a \in A} Q_\pi(s, a) \leq \max_{a \in A} Q_\pi(s, \pi(s)) \\ &= V_\pi(s) \end{aligned}$$

4.2.3 Monte Carlo Method

Monte Carlo methods learn from 'episodes' of experience with disregard to the environment, and as such it is not modelled. Instead 'learning' is performed by computing the observed mean return as an approximation of the expected return. The empirical return G_t is computed by:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t=k=1}$$

Using this to calculate the empirical mean return for state s:

$$V(s) = \frac{\sum_{t=1}^{T} (1[S_t = s] G_t)}{\sum_{t=1}^{T} (1[S_t = s])}$$

where $(1[S_t = s])$ is a binary indicator. $V(s)$ is extended to action-value by counting the (s, a) pairs:

$$Q(s, a) = \frac{\sum_{t=1}^{T} (1[S_t = s, A_t = a] G_t)}{\sum_{t=1}^{T} (1[S_t = s, A_t = a])}$$

Learning the optimal policy via Monte Carlo methods is similar to Generalised Policy Iteration however slightly different. The basic outline of the algorithm is as follows:

1. Improve Greedily $\pi(s) = \max_{a \in A} Q(s, a)$
2. Generate New Episode
3. Re-Estimate Q

4.2.4 Temporal Differences

Temporal Difference is, as Sutton and Barto 2018 describes, 'central and novel to reinforcement learning'. It is similar to Monte Carlo methods in that is is model-free however learning can come from incomplete episodes. Temporal Difference relies on a concept calling bootstrapping, whereby values are updates throughout on an ongoing process. This is done by updating $V(S_t)$ towards the 'TD Target' $(R_{t+1}+\gamma V(S_{t+1}))$. The rate of updates is controlled by the learning rate parameter α

$$
\begin{aligned}
V(S_t) &= (1-\alpha)V(S_t) + \alpha G_t \\
V(S_t) &= V(S_t) + \alpha(G_t - V(S_t)) \\
V(S_t) &= V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1} - V(S_t)))
\end{aligned}
$$

Similarly, the update function for action-value estimation:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

Now that TD Target updating is defined, TD Learning can be investigates; for that there are 3 common methods.

4.2.4.1 SARSA

SARSA refers to the order in which Q-Values are updated

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{r+1}, \ldots$$

Starting at state S_t steps are as follows:

1. Pick action, ϵ-greedy, based on Q Values : $A_t = \max_{a \in A} Q(S_t, a)$
2. Observe Reward R_{t+1}
3. Transition to State S_{t+1}
4. Pick another action $A_{t+1} = \max_{a \in A} Q(S_{t+1}, a)$
5. Update the Q Value

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \tag{4.1}$$

6. Increment t and Repeat

4.2.4.2 Q-Learning

The idea of Q-Learning was developed by Watkins and Dayan 1992 as a follow on from his Thesis 'Learning From Delayed Rewards'. It can be considered an adaptation of SARSA. Starting at S_t:

1. Pick action, ϵ-greedy, based on Q Values : $A_t = \max_{a \in A} Q(S_t, a)$
2. Observe Reward R_{t+1}
3. Transition to State S_{t+1}
4. Update the Q Value

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)) \tag{4.2}$$

The difference between Q-Learning and SARSA is updating the action value function is done by estimation of Q_*, Q-Learning forms the basis of the reinforcement learning process that is utilised in Chapter 2, Delayed Q-Learning, and as such it will be explained in more detail in Section 4.2.5

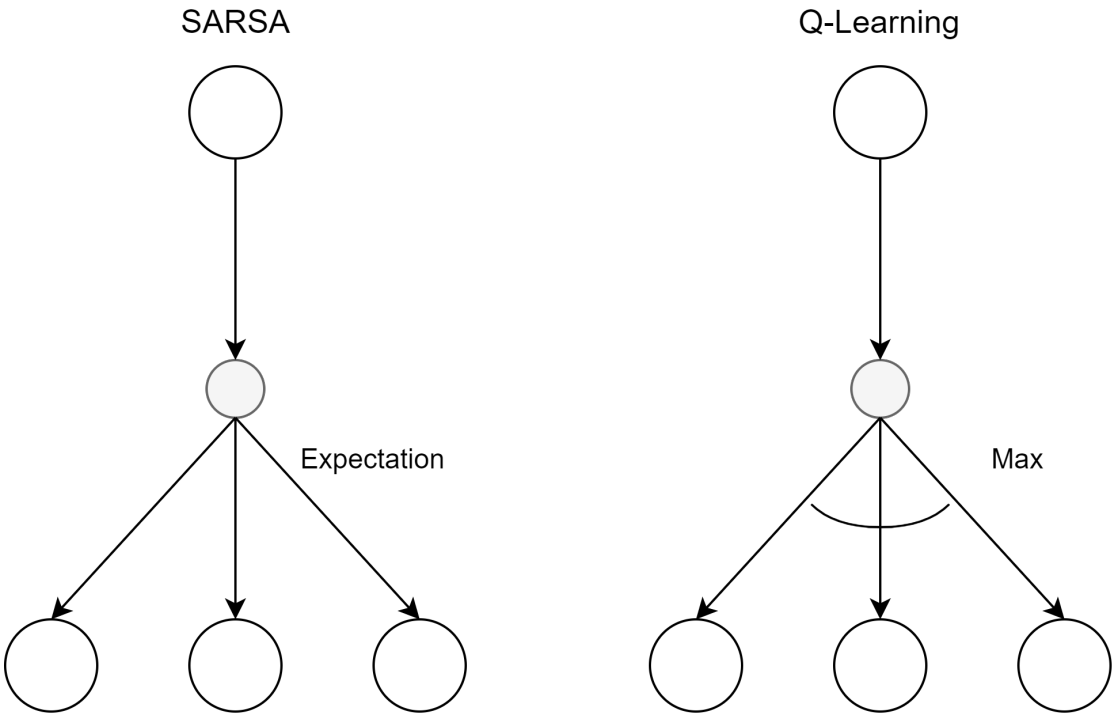


FIGURE 4.2: Flow Diagram for Q-Learning and SARSA (adapted from Figure 6.5 in Sutton and Barto 2018)

4.2.5 Delayed Q-Learning

Assuming that the function receives S,A,ϵ,δ and γ as inputs, the learning problem is defined as follows; An agent always occupies a single state of s of the MDP M. The learning algorithm is told this state requres an action, a, the agent receives a reward r and is then moved to another state s', according to the definition in Section A.3. This process repeats until a separate terminating condition is met, usually a time limit or optimality. For any fixed ϵ, Kakade 2003 states the sample complexity of exploration of an algorithm $\mathcal{A}$ to be the number of timesteps t such that the non-stationary policy at time t, $\mathcal{A}_{\sqcup}$ (which is defined by the agents history until time t)., is not ϵ-optimal from the current state, s_t, at time t ($V^{\mathcal{A}_t}(S_t) < V^*(S_t) - \epsilon$). Any algorithm $\mathcal{A}$ is then said to be PACMDP (Probably Approximatley Correct in Markov Decision Processes) if, for any ϵ and δ, the sample complexity of $\mathcal{A}$ is less than some polynomial in the relevant quantities $(S,A,\frac{1}{\epsilon},\frac{1}{\delta},\frac{1}{1-\gamma})$ with probability at least $(1-\delta)$. This definition penalises an agent for too many non-ϵ-optimal policy rather than for a non-optimal policy. Due to noise or inexperience any system may be misled about the dynamics of the system.

Therefore a failure probability of at most δ is allowed. See Kakade 2003 for full profile of this performance

Originally defined in Strehl et al. 2006. Delayed Q-Learning maintains Q-value estimates, $Q(s,a)$ for each state-action pair (s,a). At time t (discretised), let $Q_t(s,a)$ denote the algorithms current Q-value estimate for (s,a) and let $V_t(s)$ denote $\max_a Q_t(s,a)$. The agent always acts greedily with respect to its estimates, meaning that if s is the t-th state reached, $a' = \max_{a \in A} Q_t(s,a)$ is the next action chosen. In addition to Q-value estimates the algorithm has a boolean flag 'LEARN(s,a)' for each (s,a). Letting LEARN$_t(s,a)$ take the value of 'LEARN(s,a) at time t (the value before t-th action occurs). The flag indicates whether the learner is considering a modification to its Q-value estimate $Q(s,a)$. The algorithm also relies on two free parameters $\epsilon_1 \in \{0,1\}$ and a positive integer m. Finally, a counter $l(s,a)$ ($lt(s,a)$ at time t) is also maintained for each (s,a). Its value represents the amount of data (sample points) acquired for use in an upcoming update of $Q(s,a)$. Once m samples are obtained and LEARN(s,a) is true, an update is attempted.

4.2.5.1 Initialisation of the Algorithm

The Q-Value estimates are initialised to $\frac{1}{1-\gamma}$, the counters $l(s,a)$ to 0, LEARN to TRUE. That is, $Q_1(s,a) = \frac{1}{1-\gamma}$, $l_1(s,a) = 0$ and LEARN$_1(s,a)$ =TRUE for all $(s,a) \in S \times A$

4.2.5.2 Updating the Algorithm

Suppose that at time $t \leq 1$, action a is performed from state s, resulting in an *attempted update*, according to the rules defined in section 4.2.5.3. Let $s_{k_1}, s_{k_2}, \ldots, s_{k_m}$ be the m most recent next-states observed from executing (s,a), at times $k_1 < k_2 < \cdots < k_m$, respectively ($k_m = t$). For the remainder of the paper, we also let r_i denote the i-th reward received during execution of delayed Q-Learning. Therefore at time k_i, action a was taken from state s, resulting in a transition to state s_{k_i} and an immediate reward r_{k_i}. After the t-th action , the following update

occurs:

$$Q_{t+1}(s,a) = \frac{1}{m}\sum_{i=1}^{m}(r_{k_i} + \gamma V_{k_i}(s_{k_i})) + \epsilon_1 \tag{4.3}$$

as long as performing the update would result in a new Q-Value estimate that is at least ϵ_1 smaller than the previous estimate. In other words, the following equation must be satisfied for an update to occur:

$$Q_t(s,a) - \left(\sum_{i=1}^{m}(r_{k_i} + \gamma V_{k_i}(s_{k_i})\right) \leq 2\epsilon_1 \tag{4.4}$$

4.2.5.3 Using the LEARN flag

Strehl et al. 2006 provides an intuition behind the behaviour of the LEARN flags. The main computation of the algorithm is that every time a state-action pair (s,a) is experienced m times, an update of $Q(s,a)$ is attempted as in Section 4.4. For our analysis to hold, however, we cannot allow an infinite number of attempted updates. Therefore, attempted updates are only allowed for (s,a) when LEARN(s,a) is true. Besides being set to true initially, LEARN(s,a) is also set to true when any state-action pair is updated (because our estimate Q(s,a) may need to reflect this change). LEARN(s,a) can only change from true to false when no updates are made during a length of time for which (s,a) is experienced m times and the next attempted update of (s,a) fails. In this case, no more attempted updates of (s,a) are allowed until another Q-value estimate is updated.

4.2.5.4 Implementation of Delayed Q-Learning

An implementation of Delayed Q-Learning is applied in Algorithm 1

4.3 Genetic Algorithm Concepts

Genetic Algorithms (GA's) are a subset of population-based optimisation methods which are a type of Machine Learning Model, the basis is a search algorithm with a heuristic technique to mimic natural selection. GA's utilise 3 key functions; crossover, mutation and fitness. A well designed GA will often achieve optimal

Algorithm 1 Delayed Q-Learning

```
function DLQ(γ,S,A,M,ε1)
    for all (s, a) do
        Q(s, a) ← 1/(1 − γ)
        U(s, a) ← 0)
        l(s, a) ← 0
        t(s, a) ← 0
        LEARN(s, a) ← TRUE
    t* ← 0
    while LEARN(s, a) =TRUE do
        U(s, a) ← U(s, a) + r + γ max_a' Q(s', a')
        l(s, a) ← l(s, a) + 1
        if l(s, a) = m then
            if Q(s, a) − U(s, a)/m ≥ 2ε1 then
                Q(s, a) − U(s, a)/m + ε1
                t* ← t
            else if t(s, a) ≥ t* then
                LEARN(s, a) ← FALSE
            t(s, a) ← t
            U(s, a) ← 0
            l(s, a) ← 0
        if t(s, a) < t* then LEARN(s, a) ← TRUE
```

conditions, but is not guaranteed, and GA's are prone to get stuck in local maxima/minima. Genetic Algorithms population converge to a single solution. A population is made up of n phenotypes (solutions) which have m genotypes (associated properties), the fitness function evaluates all phenotypes and assigns a fitness value, the crossover function is applied to 2 (or more) phenotypes but the mutation function is applied to individual genotypes.

The intial populations are usually generated randomly, this is done to ensure maximum model variability as low variability causes sub-optimal model convergence, each population set is called Generation i. Generation $i+1$ is created by selection and mutation of Generation i, there are various selection and mutation methods that can be used, and every implementation should evaluate different function methods.

It can be useful to have a heuristic preventing some phenotypes from being selected, a common heuristic is the speciation heuristic, which penalises crossover between phenotypes that are too similar; thereby encouraging greater population

variability and thus help prevent sub-optimal convergence. Kalyanmoy Deb 1997

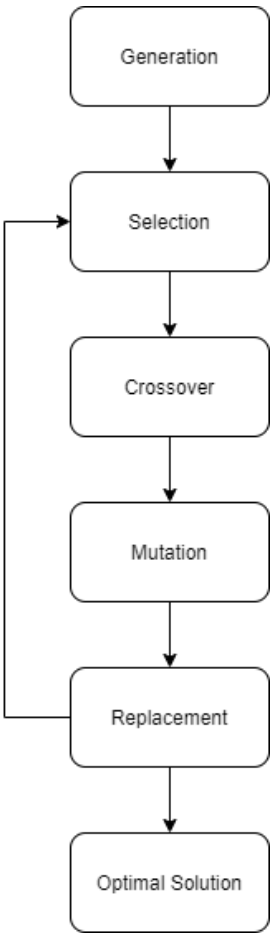


FIGURE 4.3: Flow of a Genetic Algorithm Optimisation Problem

The general idea of a GA is displayed in Figure 4.3, and Pseudo code in Listing 2. The algorithm is started by generating an initial population of chromosomes, then selecting parent chromosomes to produce two new solutions. These are then mutated (or not) and are place in a new population for the algorithm to continue until optimality or termination criteria.

4.3.1 Terminality Criteria

The algorithm created has no terminality criteria, which would cause it to never end. Some examples of termination criteria include: (Sivanandam and Deepa 2008)

1. Maximum Number of Generations,

2. Maximum Elapsed Time,

3. Not enough change, using a relative ϵ. Either by time, or by generation.

4.3.2 Chromosome Representation

Chromosomes represent a single solution, the chromosome is made up of genes (or phenotypes). In older work, chromosomes are represented as a binary string of 0s and 1s but in newer work chromosomes are often represented in matrices or strings of integers (Michalewicz 1996). The chromosome can be generated randomly, this allows for greater variability however often gives infeasible solutions. (Michalewicz 1996), explains various ways to tackle infeasible solutions:

1. Automatically deleted,

2. Left in the population to potentially create a good solution later,

3. The fitness function will deal with the infeasibility by massively penalising it,

4. Repairing the infeasibility.

Later in this chapter a combination is used, and will be explained in further detail then.

4.3.3 Population Representation

The population is effectively a matrix of chromosomes. For stability, it is important to have a large population size. Equally, a population too large leads to slow convergence. For each model it must be considered whether faster generations or larger generations are desired for optimal convergence.

4.3.4 Evaluation Function

Evaluation functions can be divided into two distinct types: minimisation and maximisation; in GA problems these are called cost and fitness respectively, and they indicate how 'good' a solution is. They are the equivalent of an objective

function in a LP problem, and for simple problems simple constraints can also be integrated into them.

4.3.5 Selection

Selection is pivotal for the success of a GA, as poor selection leads to poor new generations. Selecting good chromosomes is a complicated task and many papers deliberate on the best ways to perform them. This paper will look at some of the ones discussed in (Hopgood 2012). (It must also be noted that there are some selection algorithms that require 3+ parents, and there is discussion that this can lead to faster convergence however it is very problem dependant, and as such will not be discussed any further).

4.3.5.1 Fitness-Proportionate

A higher fitness leads to a higher likelihood of being selected, (Hopgood 2012) described an abstract roulette wheel style selection, where each chromosome has a section proportionate to their fitness value. But it also states that this method gives a good chance for low-fitness chromosomes to be selected, which is undesirable.

4.3.5.2 Fitness-Scaling

Chromosomes are ranked on a rule/function, common rules (as stated in Hopgood 2012) are linear rank scaling, non-linear rank scaling, sigma scaling, Boltzmann fitness scaling. The advantage this method gives over Fitness-Proportionate is allowing for a large search domain, helping prevent sub-optimal convergence. But is computationally inefficient as some of the rules will take significant time to calculate over 1000+ chromosomes for 1000+ generations.

4.3.5.3 Tournament

Tournament selection does not use fitness rankings, instead randomly selecting two chromosomes, and only returns the chromosome with the greatest fitness. Removing the pre-calculation of fitness or rankings makes this method far more

computationally efficient than the previous two described, however this method could potentially compare two very high fitness chromosomes, and removing one of them which is undesirable. And conversely could compare two low fitness chromosomes and keeping one of them.

4.3.6 Elitism

Elitism is a popular concept in genetic algorithms, where unlike a standard GA where the entire population is replaced with a new one, elitism prevents the best chromosomes from being deleted from subsequent generations (Hopgood 2012). In this method a replacement will not be into a new population, instead replacing the weakest chromosome in the current population. In theory this allows for faster convergence, however can minimise model variance.

4.3.7 Schemata Theorem

In order to describe further GA operators, we must first define a 'schemata', first proposed by Holland in 1960, schemata theorem is the fundamental principle explaining how evolutionary algorithms function. A **schemata** is a set of genes values that can be represented by a template (Hopgood 2012, p.177). During the

Building Blocks	is a set of schemata. The chromosome matching the schemata are said to be the instances of schema H
Order of Schemata	refers to the number of values on the fixed positions
Defining Length	is a distance between the outermost defined bits

TABLE 4.1: Definitions from Sivanandam and Deepa 2008

algorithm, the schemata is manipulated based on its relationship with the fitness value. The task of evolution is to identify those schemata and propagate them further. Good schemata tend to grow rapidly in the population due to having a short definition length and low order. (Sivanandam and Deepa 2008) state that the principles of GA's are comparable to a sequential two-armed bandit problem.

Illustrating the dilemma between exploration and exploitation (gaining new knowledge and already known knowledge respectively) which is described as a common phenomenon in modern GA practises. Equation 4.5 described the behaviour of a GA according to schemata theorem (Hopgood 2012),

$$n(H, t+1) \leq n(H, t)\frac{\bar{f}(H, t)}{\bar{f(t)}}(1 - \frac{P_c d(H)}{l-1})(1 - P_m)^{o(H)} \tag{4.5}$$

where H is the hyperplane, and variables are defined below. According to the

$d(H)$	Defining Length
l	Total Number of Phenotypes in the Chromosome
$n(H,t)$	Number of instances of H at Time t
$\bar{f}(H,t)$	Average Fitness of H
$\overline{f(t)}$	Average Fitness in population
P_c	Crossover Probability
P_m	Mutation Probability
$o(H)$	Order of H

formula defined above, the schemata with the higher average fitness will occur more frequently in future generations, and lower fitness will decrease. However this effect only occurs when P_c and P_m are low, as high crossover and mutation can be disruptive. Schemata analysis allows understanding of whether a representation is suitable for genetic and evolutionary algorithms, additionally it tells the overall efficiency, and prediction of the next generation (Sivanandam and Deepa 2008).

4.3.8 Crossover

Crossover is one of the three key concepts for genetic and evolutionary algorithms, it is the process of taking chromosomes and producing new ones, in the hope they are better. There are hundreds if not thousands of proposed crossover algorithms.

4.3.8.1 Single-Point Crossover

A random point is selected, called the 'crossover point'. Bits to the right are taken from one parent, bits to the left are taken from the other. For example, if the crossover point was 3, see Figure 4.2.

Parent 1	1	0	1	0	1	0	1	0
Parent 2	0	1	0	1	1	1	0	1
Child 1	1	0	1	1	1	1	0	1
Child 2	0	1	0	0	1	0	1	0

TABLE 4.2: Single-Point Crossover

4.3.8.2 Multi-Point Crossover

In k-point crossover, k points are randomly selected. The bits inbetween are then swapped between the parent chromosomes. This method requires more bits to be useful, as an 8-bit code with $k = 4$ produces equivalent to uniform crossover, described in the next subsection. In figure 4.3 the crossover points are $k = 2, 6$.

Parent 1	1	0	1	0	1	0	1	0
Parent 2	0	1	0	1	1	1	0	1
Child 1	1	0	0	1	1	0	0	0
Child 2	0	1	1	0	1	1	0	1

TABLE 4.3: k-Point Crossover

4.3.8.3 Uniform Crossover

In uniform crossover, each bit has an equal chance to flip, it can be extremely disruptive (and comparable to extremely high mutation), see Figure 4.4. The previously described algorithms are suitable for binary operations, but as discussed in Chromosome Representation, binary representation is rarely used due to lower readability for complex models. This leads onto PMX, PBX and OX algorithms, which are commonly used for Travelling Salesman Problem and Job Shop Scheduling where permutation representation is key.

Parent 1	1	0	1	0	1	0	1	0
Parent 2	0	1	0	1	1	1	0	1
	0.14	0.56	0.87	0.54	0.85	0.34	0.04	0.29
Child 1	1	0	0	1	1	0	0	0
Child 2	0	1	1	0	1	1	0	1

TABLE 4.4: Uniform Crossover

4.3.8.4 Partial Mapping Crossover (PMX)

Two cutting points are selected, the phenotypes inside the cutting points are copied across. The remaining phenotypes are passed according to the exchange map. To create the exchange map, you take the bits inside the cutting points. In the case of Figure 4.5 that would be $6 \rightarrow 5$, $1 \rightarrow 8$ and $4 \rightarrow 1$. Then the phenotypes from parent 2 that are **not** in the map, are copied into child 1. Where a value exists in the map, it should be replaced with the mapped value. Some papers have suggested variants on this method for k-parent methods.

Parent 1	5	3	6	1	4	7	9	2
Parent 2	2	4	5	8	1	7	6	3
Child 1	2	1	6	1	4	7	6	3
Child 2	5	3	5	8	1	7	9	2

TABLE 4.5: PMX Crossover where $k = 2, 5$

4.3.8.5 Position-Based Crossover (PBX)

PBX is a uniform crossover adapted to the permutation chromosome representation, by the addition of a binary mask. The binary mask indicates which parent's phenotype will be copied. 0 - Parent 1, 1 - Parent 2. The algorithm is designed for only 1 child, however using the inverse can create a second. See Figure 4.6

Parent 1	5	3	6	1	4	7	9	2
Parent 2	2	4	5	8	1	7	6	3
Binary Mask	0	1	0	1	1	0	1	0
Child 1	5	4	6	8	1	7	6	2
Child 2	2	3	5	1	4	7	9	3

TABLE 4.6: PBX Crossover

4.3.8.6 Order Crossover (OX)

OX is a position-based crossover modified to define a starting point and length, instead of a starting point and ending point.

Parent 1	5	3	6	1	4	7	9	2
Parent 2	2	4	5	8	1	7	6	3
Child 1	5	3	5	8	1	7	9	2
Child 2	2	4	6	1	4	7	6	3

TABLE 4.7: OX Crossover where $k = 2$ and $l = 3$

Other popular methods not discussed here include Cycle Crossover (CX), Order-Based Crossover (OX2), Voting Recombination Crossover (VR), alternating position crossover (AP) and Sequential Constructive Crossover (SCX).

4.3.9 Mutation

Mutation allows for greater exploration of the domain, and allows the initial population to not have to contain all elements of the optimal solution. Sivanandam and Deepa 2008 states that by inversion of certain phenotypes it is possible to reduce the defining length of a highly fit schema, see Eq. 4.5, which increases the population variance. Keeping a small mutation rate, keeps disruption low which is a desirable state.

4.3.9.1 Insert

Randomly selects a phenotype and reinserts it at a different position. This can cause minimal change, but could also completely change the solution depending on bit entropy.

4.3.9.2 Swap

Randomly selects 2 phenotypes and swaps them.

4.3.9.3 Inversion

Randomly selects an arbitrary subsection in the chromosome and reverses the order of the phenotypes and reinserts them in the reversed order.

4.3.9.4 Scramble

Randomly selects an arbitrary subsection in the chromosome and randomly rearranges the phenotypes.

4.3.10 Crossover and Mutation Values

Crossover and Mutation functions have extremely different purposes, and both are required. There are methods of determining genetic parameters

1. Manual Experimentation, by performing the algorithm with values
2. Manual Optimisation, calculating by hand
3. Evolving Operators (Hopgood 2012)

Algorithm 2 Genetic Algorithm

```
P ← generatePopulation[100]          ▷ Get 100 random variables
while endCondition = false do
    P ← selection(P)
    P ← crossover(P)
    P ← mutation(P)
    B ← fitness(P)
    if B == optimal then
        endCondition ← true
     return B,P
```

Chapter 5

Genetic Algorithms Application

5.1 1-0 Knapsack Application

Genetic Algorithms are effective for solving potentially complex ILP Problems, here is a demonstration of the common 1-0 knapsack problem solved using a GA Approach.

5.1.1 Model Definition

$$\begin{aligned} \underset{x}{\text{maximise}} \quad & \sum_{i=1}^{n} x_i v_i \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i w_i \leq W, \\ & x_i \in 0, 1, \ i \in n. \end{aligned} \tag{5.1}$$

Where x_i is the decision variable of whether object i is being taken or not, v_i and w_i are the objects associated value and weight. Informally the objective is the maximise value, without exceeding the weight limit.

5.1.2 GA Definition

Initial Pop : Random w_i and v_i values

Fitness : if $\sum x_i w_i$ then $\sum x_i v_i$

Crossover : Single Point Crossover

Mutation : Random Chance

5.1.3 Implementation

Fitness evaluation is key for the success of a model, Listing 5.1

```
def cal_fitness(weight, value, population, threshold):
    fitness = numpy.empty(population.shape[0])
    for i in range(population.shape[0]):
        S1 = numpy.sum(population[i] * value)
        S2 = numpy.sum(population[i] * weight)
        if S2 <= threshold:
            fitness[i] = S1
        else :
            fitness[i] = 0
    return fitness.astype(int)
```

LISTING 5.1: $1-0$ Knapsack Genetic Algorithm: Fitness Function

Crossover evaluation is key for the success of a model, Listing 5.2

```
def crossover(parents, num_offsprings):
    offsprings = numpy.empty((num_offsprings, parents.shape[1]))
    crossover_point = int(parents.shape[1]/2)
    crossover_rate = 0.8
    i=0
    while (parents.shape[0] < num_offsprings):
        parent1_index = i%parents.shape[0]
        parent2_index = (i+1)%parents.shape[0]
        x = rand.random()
        if x > crossover_rate:
            continue
        parent1_index = i%parents.shape[0]
        parent2_index = (i+1)%parents.shape[0]
```

```
14          offsprings[i,0:crossover_point] = parents[parent1_index,0:
    crossover_point]
15          offsprings[i,crossover_point:] = parents[parent2_index,
    crossover_point:]
16          i=+1
17      return offsprings
```

LISTING 5.2: $1-0$ Knapsack Genetic Algorithm: Crossover Function

Mutation evaluation is key for the success of a model, Listing 5.3

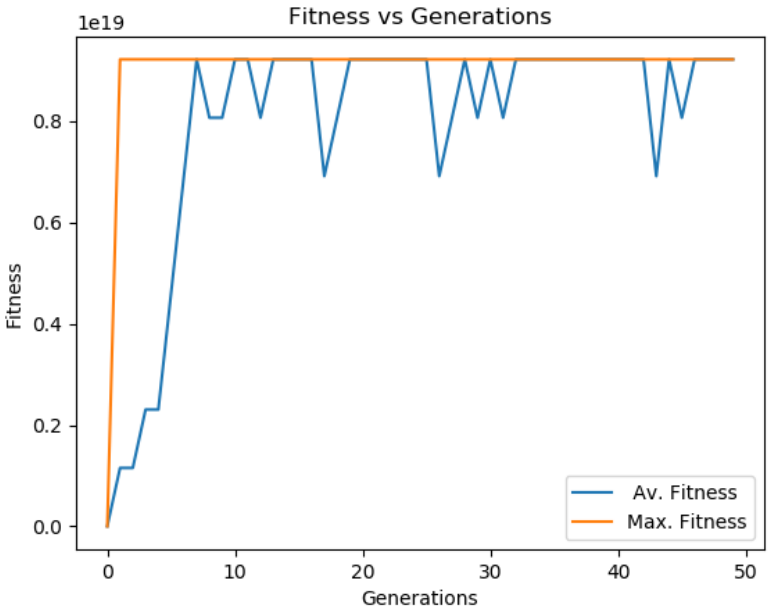
```
1  def mutation(offsprings):
2      mutants = numpy.empty((offsprings.shape))
3      mutation_rate = 0.4
4      for i in range(mutants.shape[0]):
5          random_value = rand.random()
6          mutants[i,:] = offsprings[i,:]
7          if random_value > mutation_rate:
8              continue
9          int_random_value = randint(0,offsprings.shape[1]-1)
10         if mutants[i,int_random_value] == 0 :
11             mutants[i,int_random_value] = 1
12         else :
13             mutants[i,int_random_value] = 0
14     return mutants
```

LISTING 5.3: $1-0$ Knapsack Genetic Algorithm: Mutation Function

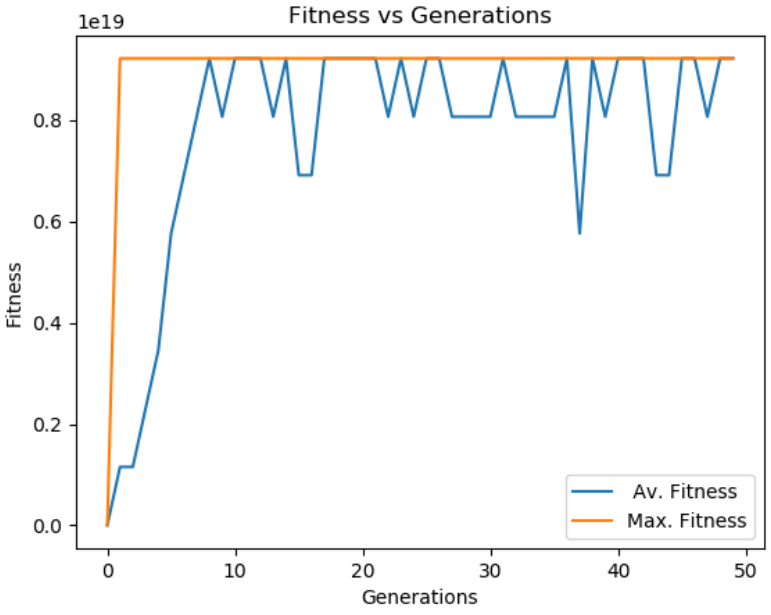
5.1.4 Results

Since the 1-0 knapsack problem is a very simple problem, we are using a population size of 8 and a terminating condition of Maximum 50 Generations. There are 30 possible items, each with an assigned weight and value. The maximum weight is 35;

It can be seen from Figure 5.1 that each time the program is run, a different path is taken to optimality, each time the Maximum fitness is maximum by the 3rd generation however the Average fitness of a generation is never able to achieve full



(A) First Run



(B) Second Run

FIGURE 5.1: knapsack.py Fitness across Generations

optimality, this is due to the mutation allowing for changes. This is good as it could prevent sub-optimal convergence of the model.

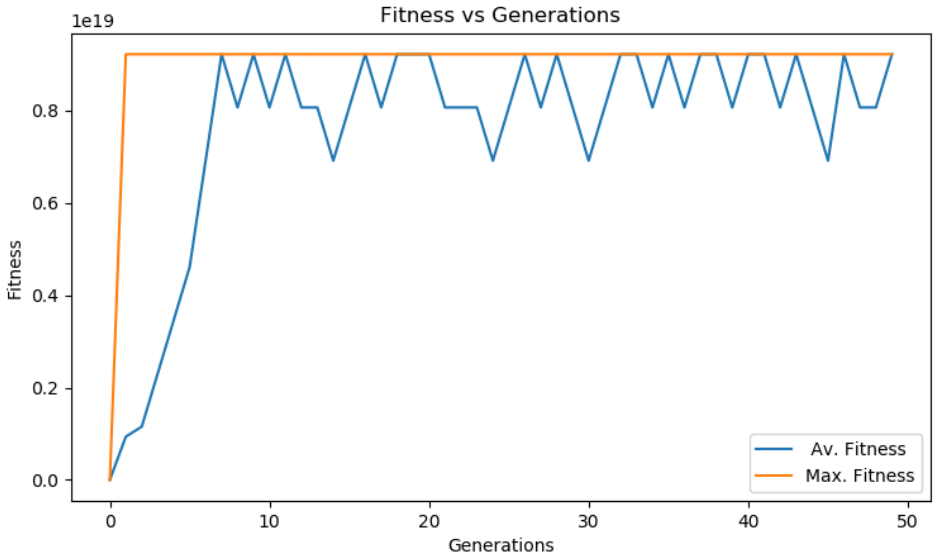


FIGURE 5.2: knapsack.py Fitness across Generations : Third run

5.2 TSP Problem Description

The same principles applied to the 1-0 Knapsack problem in Section 5.1, can be applied to common railway scheduling problems, this section will define a simple timetabling problem, and a genetic algorithm to optimise it.

5.2.1 Network Description

Define a network of nodes, n, their associated distances from all other nodes, matrix N_d, their associated dwell times, n_w.

5.2.2 Model Description

The phenotypes are defined as P_i (generally, a population size of 100 is used in these experiments), to allow for mutation the genotypes are represented as binary bits in a json-style object (defined in Listing 5.4).

```
phenotype = {
    toc: 00                     // 2 bits, 4 values
    line: 00000000              // 8 bits, 256 values
    unit: 00000000              // 8 bits, 256 values
    origin:0000000000           // 12 bits, 4096 values
    destination:0000000000      // 12 bits, 4096 values
```

```
7       intermediate = []
8   }
9
10  intermediate = {
11      #station# = {
12          arrival = 0000000000     // 12 bits, 4096 values
13          departure = 0000000000   // 12 bits, 4096 values
14      }
15  }
```

Listing 5.4: Phenotype Variable

Where intermediate is a sub-object containing any number of stations, which is known by the line variable in the phenotype. Times are all represented as 12 bits, this is to model them as times of day, since there are 1440 minutes we can model times in half-minute intervals. It is important for computer memory to be a consideration, due to each generation have 100's or 1000's of phenotypes at a time.

5.3 Toy Scenario

A very simple 5 station, 2 bi-directional track, see Figure 5.3. The stations are denoted A,B,C,D,E and F. A mutation rate of 0.05% will be used.

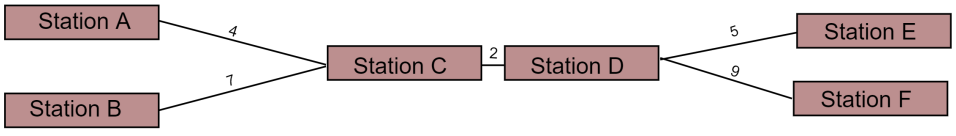


Figure 5.3: Toy Example Network Layout

Initial Pop : Random Bits

Fitness :

Crossover : k- Point Crossover

Mutation : Random Chance

In Figure 5.4 the different timetable components are broken down for the $A \rightarrow F$ section of the timetable. Traditionally timetables are presented directionally, and that will also occur in this report. In Figure 5.5 the opposing timetable is shown, the only factors at play are arc lengths, in this example it is measured as time, however in further work it could be distance where time is calculated based on a train speed ODE calculation. (Figures 5.4, 5.5 and 5.6 all have conditions $s = \left(\begin{smallmatrix} 0 & 2 \\ 0 & 2 \end{smallmatrix}\right)$, $d = 1$ where s is Starting time and d is Dwell time.)

5.3.1 Decoding Tables

For definitions of objects used see Listing 5.4

Encoded Bits	Decoded Meaning
TOC	
00	Reserved
01	MMU Rail
10	Un-Used
11	Un-Used
Line	
00000000	Reserved
00000001	A to F
00000010	A to E
00000011	B to F
00000100	B to E
00000101	F to A
00000110	F to B
00000111	E to A
00001000	E to B
Unit	
00000000	Reserved
00000001	BR Class 331
00000010	BR Class 669

TABLE 5.1: Toy Example Decode Table

The representation of s becomes far more important, in the basic example, for the optimised version, where a cyclic timetable is required (a train must start where another ends). Figure 5.6 is the combination of Figures 5.4 and 5.5. It can be seen that there are a number of intersections where problems could occur, in this version of the example there would be no problems due to an unlimited number

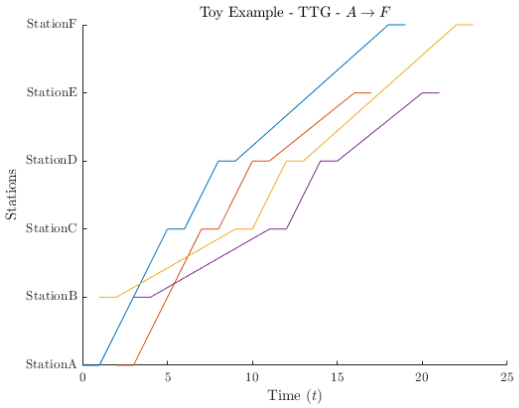


FIGURE 5.4: Toy Example AB → EF

of trains (and also the two parallel rails, as in the problem definition). Then optimisation must be applied, as shown in Figure 5.7. The limit was imposed at 3 trains, as this seemed a proportionate number of trains for the network layout (having 4 terminals and 2 pass-throughs). The lines are multi-coloured so that it is still obvious where each schedule entry changes. It is also clear what a specific rolling stock would be doing at any point. Additionally, if the model allowed it,

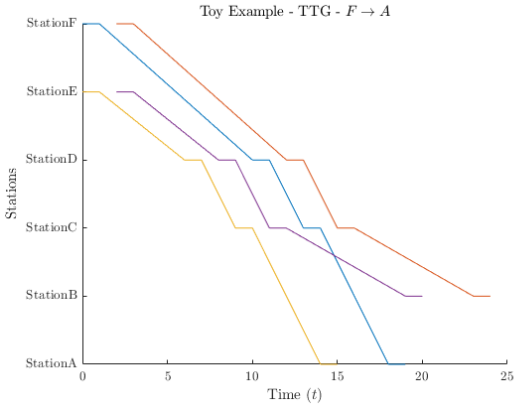


FIGURE 5.5: Toy Example EF → AB

this optimal solution could be made better by modelling over a further period as it would allow for more flexibility and less downtime, as it can be seen that one of the trains will stop at 40 minutes then wait for the next hour to begin again. However, although an unintended outcome, this could be interpreted as an extended break for the workers or downtime for the train due the fact that over $3T$ time periods, each train will have done each route. This model is not perfect,

but since it is just the most basic of examples, it will not be re-worked, but it gives a good understanding of what is going on to the reader.

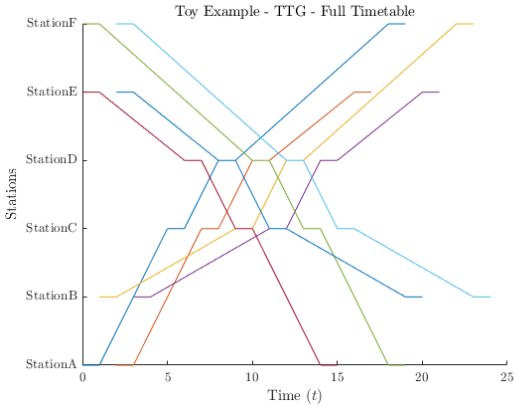


FIGURE 5.6: Toy Example Combined

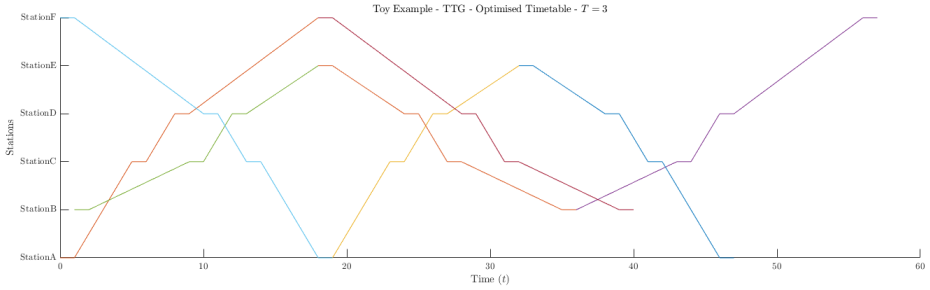


FIGURE 5.7: Toy Example Combined Optimised

Dwell has a large impact on robustness of timetables as can be seen in Figure 5.8 where the previously feasible timetable has become infeasible, it is important to minimise downtimes and delays in the railway as knock on effects occur easily.

5.3.2 Case Study

To understand the uses of genetic algorithms in railway scheduling they must be applied to a real life example, here the Wigan to Manchester lines are chosen (see Figure 5.9). This network provides 2 distinct routes from Area A (Wigan) to Area B (Manchester), currently the Northern Rail franchise agreement is to operate 4tph to Manchester Victoria and 1.5tph to Manchester Picadilly. 3 of these go via Walkden (Route B) and the others go via Bolton (Route A), except the xx:23 which goes via a different route stopping only at Wigan then Manchester. Other distinctions are stopping locations, so not all trains stop at Ince, Moses Gate,

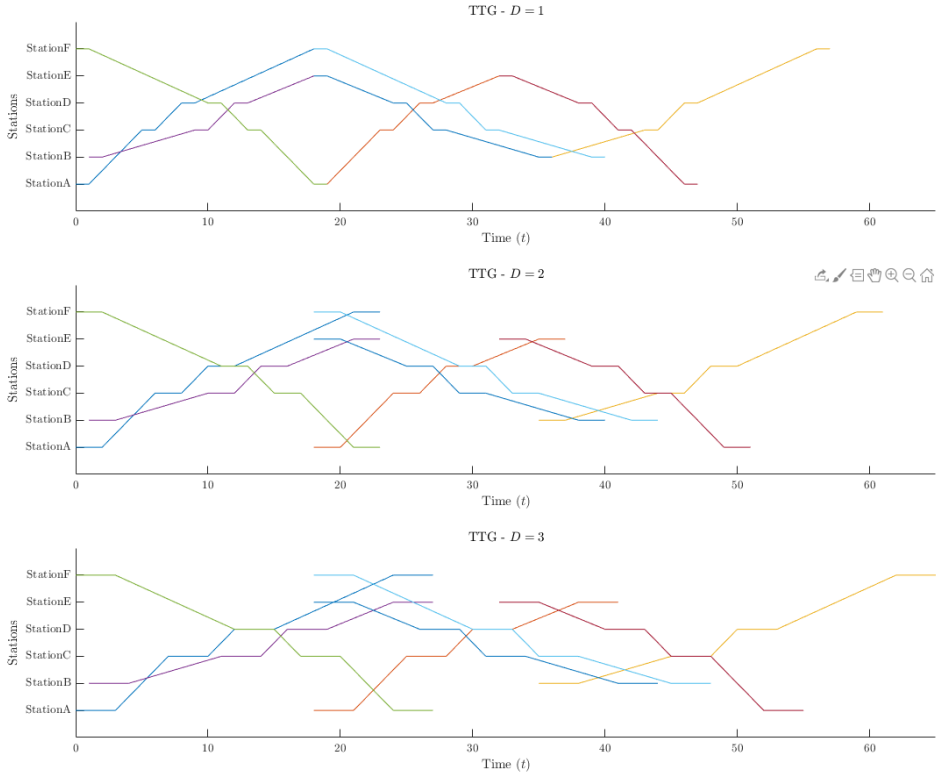


FIGURE 5.8: Toy Example Varying Dwell

Kearsley, Farnworth and Clifton on Route A. And not all trains stop at Daisy Hill, Hag Fold or Moorside on Route B. Therefore each stopping pattern requires its own route, which can be seen in the decode table

An initial solution is shown in Figure 5.10. This is a single direction representation of 4 trains, 2 on each route

5.4 Conclusions

In this chapter the theory of genetic algorithms was introduced and implemented on a 1-0 knapsack problem and a basic TSP, further work would be converting the problem defined in Chapter 5 into a genetic algorithm as this is a very complex problem that could benefit from this style of optimisation.

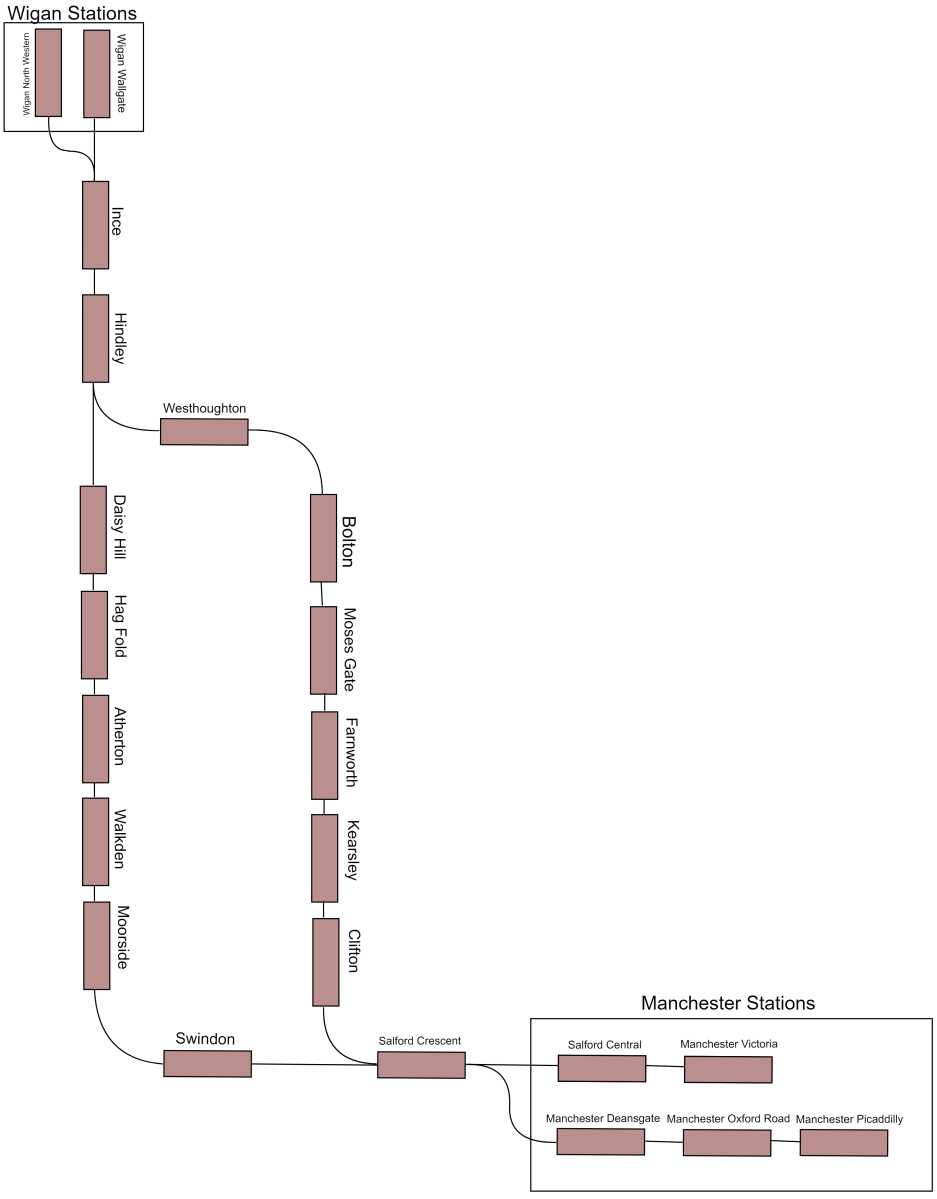


FIGURE 5.9: Wigan → Manchester Lines

Encoded Bits	Decoded Meaning
TOC	
00	Reserved
01	Northern Rail
10	TransPennine Express
11	Un-Used
Line	
00000000	Reserved
00000001	Wigan to Manchester : Route A : Stopping All
00000010	Wigan to Manchester : Route A : Stopping Reduced
00000011	Wigan to Manchester : Route B : Stopping All
00000100	Wigan to Manchester : Route B : Stopping Reduced
00000101	Manchester to Wigan : Route A : Stopping All
00000110	Manchester to Wigan : Route A : Stopping Reduced
00000111	Manchester to Wigan : Route B : Stopping All
00001000	Manchester to Wigan : Route B : Stopping Reduced
Unit	
00000000	Reserved
00000001	BR Class 142 : 2 Carriages
00000010	BR Class 144 : 2 Carriages
00000011	BR Class 144 : 3 Carriages
00000101	BR Class 150 : 2 Carriages
00000110	BR Class 153 : 1 Carriages
00000111	BR Class 155 : 2 Carriages
00001000	BR Class 156 : 2 Carriages
00001001	BR Class 158 : 2 Carriages
00001010	BR Class 158 : 3 Carriages
00001011	BR Class 170 : 3 Carriages
00001100	BR Class 180 : 5 Carriages
00001101	BR Class 195 : 2 Carriages
00001110	BR Class 195 : 3 Carriages
00001111	BR Class 319 : 4 Carriages
00010000	BR Class 322 : 4 Carriages
00010001	BR Class 323 : 4 Carriages
00010010	BR Class 331 : 3 Carriages
00010011	BR Class 331 : 4 Carriages
00010100	BR Class 333 : 4 Carriages

Table 5.2: Small Example Decode Table

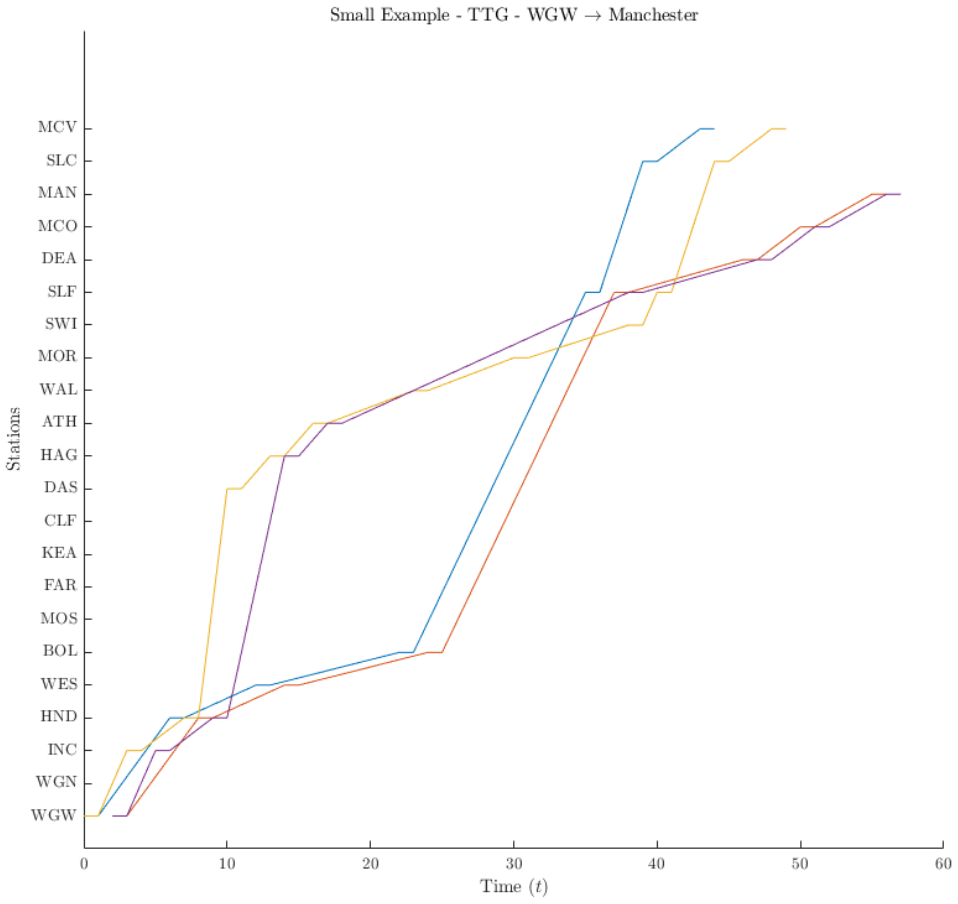


FIGURE 5.10: Wigan → Manchester Time Graph

Chapter 6

Reinforced Learning Application for Train Scheduling Problem

6.1 Introduction

This chapter introduces an MILP model and a discrete event simulator with reinforced learning techniques. Large sections of program logic are based on a program by github user iquadrat (*Best Solver* 2018).

6.2 Model Formulation

Initially the problem is modelled as an MILP, as in this section, which is used to help model the program defined in Section 6.3.

6.2.1 Data Sets

The input data files contain Service Intentions (a list of trains to be scheduled, and a description of routes it could take), Routes (modelled as a directed acylic graph, it provides list of possible ways to navigate the railway network), Resources (models blocking and following logic) and parameters for solver-specific guidelines. The Route object is an id and a list of route paths, which are made up of route sections which are described in Table 6.1.

JSON Name	Model Name	Description
sequence_number		Order Number, The train passes over the sections in this order
penalty	p	To deter trains from using specific sections
starting_point		Used for visualisations
ending_point		Used for visualisations
minimum_running_time	mr	Minimum Duration a train must spend in a section
section_markers		Labels that mark this section as a potential section to fulfil a requirement
resource_occupations		
resource		A reference to the id of the resource that is occupied
occupation_direction		A description of the direction in which the resource is occupied.

TABLE 6.1: Input Data Modelled Variables : Route

In addition to these parameters defined in the input model, there are also some set definitions required (see Table 6.3). Using the parameters defined in the input data model, and the definitions defined above it is possible to model constrains, variables and an objective function.

6.2.2 Variables

There are two superscripts used in the formulation documentation:

1. *entry*, indicates a variable relates to the entry of a section
2. *exit*, indicates a variable relates to the exit of a section

Subscripts are also used throughout, indicating an index of an array or set.

For each service intention occupying a resource of a route:

$$t^{entry}_{s,r,rs} - \text{Entry Time of Service Intention } s \text{ into section } rs \text{ of route } r \tag{6.1}$$

$$t^{exit}_{s,r,rs} - \text{Exit Time of Service Intention } s \text{ into section } rs \text{ of route } r \tag{6.2}$$

JSON Name	Model Name	Description
sequence_number		Order Number
type		Description what the requirement is
min_stopping_time	*ms*	
entry_earliest	*inEar*	
entry_latest	*inLat*	
exit_earliest	*outEar*	
exit_latest	*outLat*	
entry_delay_weight	*weightIn*	
exit_delay_weight	*weightOut*	
connections		Allows modelling connections
onto_service_intention		Reference to the *service_intention* that accepts the connection
onto_section_marker		Specifies which *route_sections* in the *onto_service_intention* are candidates to fulfil the connection
min_connection_time		minimum duration required between arrival and departure event

TABLE 6.2: Input Data Modelled Variables : Service

These variables handle representation of a timetable with an assigned route of section times. There are 3 binary variables that must be defined that each serve a different purpose: Equation 6.3 ensures a unique route is selected for each train.

$$\alpha_{s,r} = \begin{cases} 1 & \text{if Service Intention } s \text{ uses route } r \\ 0 & otherwise, \end{cases} \tag{6.3}$$

Equation 6.3 ensures the route resources are allocated correctly

$$\beta_{s,e} = \begin{cases} 1 & \text{if Service Intention } s \text{ uses resource } e \\ 0 & otherwise, \end{cases} \tag{6.4}$$

Equation 6.5 ensures section collisions are handled correctly, and ensures delay is applied correctly

$$\gamma_{s_1,s_2,e} = \begin{cases} 1 & \text{if both Service Intention } s_1 \text{ and } s_2 \text{ use the resource } e \\ 0 & otherwise, \end{cases} \tag{6.5}$$

Set ID	Description
$\mathcal{S}$	Set of Service Intentions, indexed by S_s
$\mathcal{P}$	Set of Paths of each S, indexed by $P_{s,r}$ where S is the Service Intention and r is the desired route
$\mathcal{R}$	Set of Route Sections of each Path of each Service Intention, indexed by $R_{s,r,rs}$ where rs is the route section specified
$\mathcal{E}$	Set of Resources to be used by each Service Intention. Indexed by $E_{s,e}$ where e is the resource specified
$\mathcal{L}$	Set of Resources of each Service Intention at each Path. Indexed by $L_{s,r,e}$
$\mathcal{I}$	Set of Service Intention Pairs, and possible resources occupied by them. Indexed by $I_{s1,s2,e}$ where $s1$ is Service Intention 1 and $s2$ is Service Intention 2
$\mathcal{F}$	Set of Service Intention Paris, their associated routes and resources. Indexed by $F_{s1,r1,s2,r2,e}$ where $r1$ is the Route associated with $s1$ and $r2$ is the Route associated with $s2$

TABLE 6.3: Input Data Modelled Variables : Sets

6.2.3 Constraints

For a model to give useful results, it must be modelled accurately and to do this constraints are used to eliminate infeasible solutions. The first constraint is displayed in Equation 6.6 and it states that if $\alpha_{s,r} = 1$ (if r is the selected path of a Service Intention s) the minimum time between entering a section and leaving a section is the sum if the minimum running time and minimum stop time.

$$\alpha_{s,r} = \begin{cases} 1 & t^{exit}_{s,r,rs} - t^{entry}_{s,r,rs} \leq mr_{s,r,rs} + ms_{s,r,rs} \\ 0 & nothing, \end{cases} \tag{6.6}$$

This constraint, in its current form, cannot be applied so it must be linearised, as displayed in Equation 6.7. Using the Big M LP Method and assuming that $M \leq mr_{s,r,rs} + ms_{s,r,rs} + t^{entry}_{s,r,rs}$.

$$t^{exit}_{s,r,rs} - t^{entry}_{s,r,rs} \geq mr_{s,r,rs} + ms_{s,r,rs} - M(1 - \alpha_{s,r}) \; \forall \; [s, r, rs] \in \mathcal{R} \tag{6.7}$$

The second constraint is displayed in Equation 6.8; this constraint keeps the sequence of times along a route, 6.9 ensures that the correct section is blocked, and

that as soon as one section is exited, it enters the next section.

$$t^{entry}_{s,r,rs} \leq t^{exit}_{s,r,rs} \qquad \forall\ [s,r,rs] \in \mathcal{R} \tag{6.8}$$

$$t^{exit}_{s,r,rs} \leq t^{entry}_{s,r,(rs+1)} \qquad \forall\ [s,r,rs] \in \mathcal{R} \tag{6.9}$$

The third and fourth constraint, Equations 6.10 and 6.11, state the requirements for the intended route of each potential train, Equation 6.10 prevents a train departing earlier than the scheduled departure and equation 6.11 prevents a train from setting off too early at a station; accurately modelling passenger deboarding/boarding. These constraints only apply if $\alpha_{s,r} = 1$, if service intention s uses route r.

$$t^{entry}_{s,r,rs} \geq inEar_{s,r,rs} \qquad \forall\ [s,r,rs] \in \mathcal{R} \tag{6.10}$$

$$t^{exit}_{s,r,rs} \geq outEar_{s,r,rs} \qquad \forall\ [s,r,rs] \in \mathcal{R} \tag{6.11}$$

The fifth and sixth constraint, Equation 6.12 and 6.13, allocates exactly one path to each train and associates the relationship respectively.

$$\sum_{r \in P} \alpha_{s,r} = 1 \qquad \forall\ s \in \mathcal{S} \tag{6.12}$$

$$\beta_{s,e} \geq \alpha_{s,r} \qquad \forall\ [s,r,e] \in \mathcal{RSE} \tag{6.13}$$

The seventh constraint, displayed in Equation 6.14, handles train interactions. It applies where $\alpha_{s_1,r_1} = \alpha_{s_2,r_2} = 1$ and $e \in \{r_1, r_2\}$. The binary variable, γ, determines that when two trains want to occupy a section, the first train to attempt to enter is the only train that can enter.

$$t^{entry}_{s_1,r_1,rs} - t^{entry}_{s_1,r_1,rs} \begin{cases} \leq 0 & \gamma_{s_1,s_2,e} = 1 \\ = 0 & nothing, \\ \geq 0 & \gamma_{s_1,s_2,e} = 0 \end{cases} \tag{6.14}$$

Following on from constraint 7 (Equation 6.14), constraint 8 states that the value

of γ determines the delay that must be incured on the train that waits to avoid a collision, and as such it only applies where $\alpha_{s_1,r_1} = \alpha_{s_2,r_2} = 1$ and $e \in \{r_1, r_2\}$.

$$\beta_{s_1,s_2,e} = \begin{cases} 1 & t^{entry}_{s_2,r_2,e} \leq t^{exit}_{s_1,r_1,rs} + R \\ 0 & t^{entry}_{s_1,r_1,e} \leq t^{exit}_{s_2,r_2,rs} + R \end{cases} \tag{6.15}$$

However constraints 7 and 8 cannot be applied, and as such are linearised using big M. Equation 6.16 fixes $\gamma_{s_1,s_2,e} = 1$, which is where s_1 enters before s_2 and equation 6.17 fixes $\gamma_{s_1,s_2,e} = 0$, which is where s_2 enters before s_1. $(\forall\{s_1, r_1, s_2, r_2, e\} \in F)$

$$t^{entry}_{s_1,r_1,e} - t^{entry}_{s_2,r_2,e} \geq M(1 - \beta_{s_1,s_2,e}) + M(2 - \alpha_{s_1,r_1} - \alpha_{s_2,r_2}) \tag{6.16}$$

$$t^{entry}_{s_2,r_2,e} - t^{entry}_{s_1,r_1,e} + \epsilon \geq M\beta_{s_1,s_2,e} + M(2 - \alpha_{s_1,r_1} - \alpha_{s_2,r_2}) \tag{6.17}$$

Equation 6.18 states that where there is double-occupation and $\gamma_{s_1,s_2,e} = 1$, s_2 must be delayed until s_1 has exited, with headway R. Equation 6.19 is the reverse as described above.

$$t^{exit}_{s_1,r_1,e} - t^{entry}_{s_2,r_2,e} + R \geq M(1 - \beta_{s_1,s_2,e}) + M(2 - \alpha_{s_1,r_1} - \alpha_{s_2,r_2}) \tag{6.18}$$

$$t^{exit}_{s_2,r_2,e} - t^{entry}_{s_1,r_1,e} + R \geq M\beta_{s_1,s_2,e} + M(2 - \alpha_{s_1,r_1} - \alpha_{s_2,r_2}) \tag{6.19}$$

6.2.4 Objective Function

The objective function is to minimise total train delay, where a delay is a deviation between realised times and scheduled times of arrival or departure of a station or section. Thus the measure of a successful timetable is the weighted sum of all train delays, if no train is late then the delay is zero, therefore the objective function is 0. In addition to this there are 'penalty routes' whereby trains are discouraged by taking them, so even if a train is on time, by taking the route the train will incur

a penalty which increases the objective value.

$$\frac{1}{60} \times \left[\sum_{\mathcal{S},\mathcal{R},\mathcal{RS}} weightIn_{rs} \times \max\left(0, \left(t^{entry}_{s,r,rs} - inLat_{s,rs}\right)\right) + weightOut_{rs} \times \max\left(0, \left(t^{exit}_{s,r,rs} - outLat_{s,rs}\right)\right)\right] + \sum_{\mathcal{S},\mathcal{R},\mathcal{RS}} p_{s,rs} \times \beta_{s,rs} \quad (6.20)$$

Term 1 of Equation 6.20 is the weighted summation of the difference between the entrance time and latest entry time in all sections, plus the difference between the current exit time and latest exit time. Term 2 of Equation 6.20 includes all route penalties. The sum of Term 1 and Term 2 is averaged over a 60 minute period to give the final objective function value.

6.3 Solver

The model defined in the previous section, 6.2, is a mixed integer optimisation problem which is NP-hard, the constraints make it difficult/impossible to solve exactly in any time, let alone a reasonable amount of time. In order to solve this problem, a custom simulator package was designed to make a 'good-enough' solution. The optimisation program, named tSim, is a discrete event simulation program and is created in python and utilises several packages. Key packages are json, os, sys and numpy. Additionally time is used to comply with iso-datetime formats. Python runtime variables are used to configure the random seed, in addition to the problem that is to be solved. The core of the solver is displayed in Figure 6.1 and ensures that all service intentions get scheduled, all blockages are handled as events in the simulator class. The simulator attempts to schedule trains one section at a time on a FIFO basis. Each step, the simulator looks at where the train currently is on the route, and checks whether it can proceed. If not, it waits a fixed number of seconds. If it can proceed the next route section is chosen ϵ-greedily among all possible sections. How desirable the route sections

are is encoded in a table, qTable. These values are initialised and then updated during the simulation session. Choices that lead to delays are penalised.

6.3.1 Discrete Event Simulation

An instance of tSim holds all required information of a scenario including multiple trains, resources and timetables. tSim uses discrete event simulation to schedule the service intentions. Within the tSim class, there are various functions that perform time stepping and controlling the time stepping. The *run* function, psuedo-code in Algorithm 3, is called every loop of the handler script, flowchart of the handler script in figure 6.1, this is to improve optimal values. The events are handled in order, events are differentiated and scheduled in the *runNextTrain* function, psuedo-code in Algorithm 3. This python program is custom purpose, however it is based on the *simPy* package.

Algorithm 3 tSim Control Functions

```
procedure RUN(self)
    t ←minTime
    while t < (maxTime+10) do
        for e in event[t] do
            runNextTrain[e]
        t ← t + 1
    return calculateScore()
procedure RUNNEXTTRAIN(self,e)
    if e = type(Node) then runNode()
    else if e = type(Resource) then del e.resource
    else if e = type(Station) then nextEvent ← newevent[e]
```

Algorithm 4 tSim Node Function

```
procedure RUNNODE(self,e)                    ▷ Entering a Node
    checkEligible()          ▷ Check for all termination conditions
    s ← q.getStateID
    l ← q.getLink
    update self                              ▷ Update Solution
```

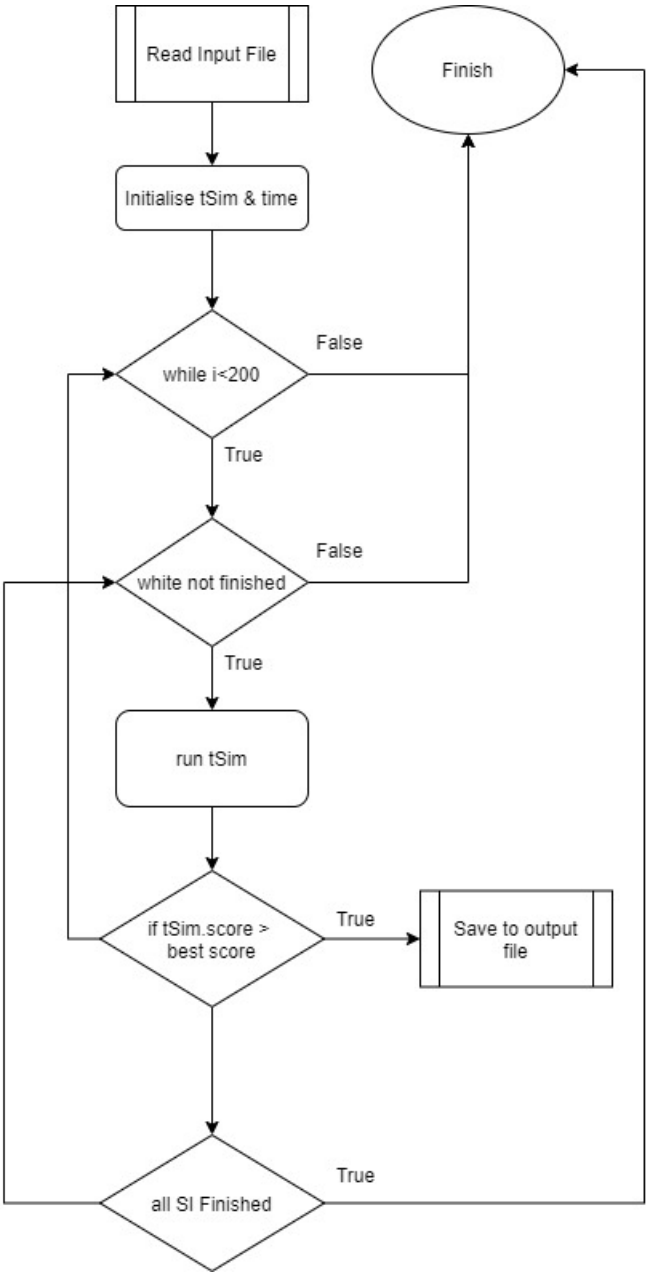


FIGURE 6.1: tSim Handler Script Flowchart

6.3.2 Delayed Q-Learning Implementation

The reinforcement learning algorithm used is Delayed Q-Learning, it is described in detail in Section 4.2.5, the Q-Learning setup is defined in the handler script, a tSim.QTable instance is created (then a tSim instance is initialised before QTable args), and given $\epsilon = 0.1, \delta = 0.8, \gamma = 0.8$, where ϵ is the optimality factor, δ is the learning rate and γ is the discount factor. In the tSIm.QTable instance is a dictionary of Q-Values, each with an associated dictionary of 'options', initially

these are all random (if they are 0, the algorithm does not work, so they must be initialised as random numbers) and are updating by the update function, pseudo-code in Algorithm 5. Calls for state-actions are made to the *getAction* function, psuedo-code in Algorithm 5, which returns the optimal action to make.

Algorithm 5 tSim.QTable Class Functions

```
procedure GETACTION(self,o,s)        ▷ Getting Action for any option or state
    if length(o) = 1 return o[0] then
    if rand¡ϵ then                   ▷ Uniform Random Package, for better convergence
        a ← rand(o)
    else
        m ← −∞                                          ▷ Maximum Q-Value
        a ← null                                        ▷ Action to be performed
        for all o do
            v ← qval(s, o[i])        ▷ qval is predicted reward of action o[i]
            if v > m then
                a ← c
                m ← v
    return a
procedure UPDATE(self,S_n,S_{n−1},A,r)                  ▷ Updating Q-Values
    p ← qval(S_{n−1}, A)                                ▷ Previous Value
    m ← 0
    if qval(S_n) > 0 then                               ▷ If exists, update max
        m ← max(qval(S_n))
    v ← (1 − α) ∗ p + α ∗ (r + γ ∗ m)                   ▷ New Value
    return v                         ▷ Update Q-Table with new Q-Value
```

6.4 Simple Scenario

To fully understand the flow of the program it can be useful to run through an example, this simple scenario is based on that provided by *Train Schedule Optimisation Challenge* 2018. In Table 6.4 the minimum times are provided and labelled either *eIn*, *eOut* and *lOut* which represents earliest entry, earliest exit and latest exit respectively. The scenario is modelled as a directed acyclic graph in Figure 6.2a with route 111 highlighted in blue and route 113 highlighted in red. It must be noted that whilst service 111 stops at station B, service 113 does not; this is detailed by the fact that it has an earliest exit time. Service 111 starts from section *A*3 which can be interpreted as Station A Platform 3, Service

113 starts from section $A1$ which can be interpreted as Station A Platform 1. Station B is a single platform station and as such only have one section, Station C has 2 platforms with very different paths to them, this could be thought of as a subsection of a busy multi-platform terminus whereby there are various tracks converging/diverging. Nevertheless, both services use Station C Platform 1, where both services terminate, and this scenario is over. Node calling times are calculated

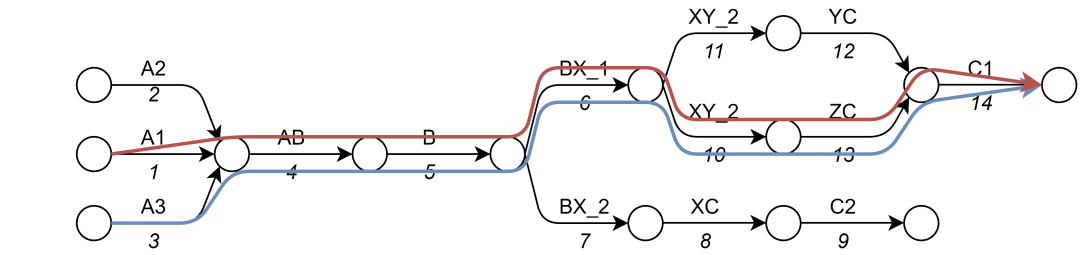
Train	A	B	C
111	eIn 08:20:00	eOut 08:30:00	lOut 08:50:00
113	eIn 07:50:00		lOut 08:16:00

TABLE 6.4: Sample Scenario Service Plan

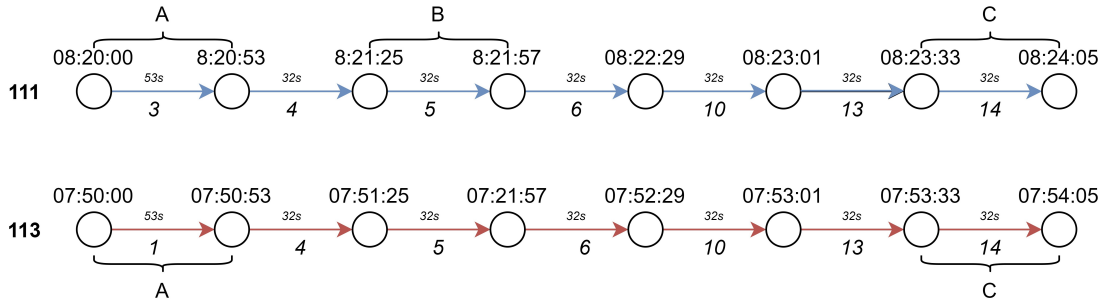
using the $n_{i-1} + t$ where n_{i-1} is the calling time of the previous node and t is the travel time between the nodes. Unless waiting is involved, in this scenario a hard constraint is broken which requires service 111 to leave Station B at 08:30:00 earliest, in the original schedule (Figure 6.2b) this rule is broken and therefore infeasible, this is rectified in Figure 6.2c where the new times are in red. This problem was solved using the tSim program as described in Section 6.3 in addition to a MILP as described in Section 6.2. This sample scenario is the most complex the MILP model is able to solve due to complexity, time to solve became above 5000s for all other problems. The computation times are displayed in Figure 6.3, Solver took 0.04 seconds; of which most will have been reading and writing files where as the MILP took 10.65 seconds in Gurobi (*Gurobi* 2020).

6.5 Results

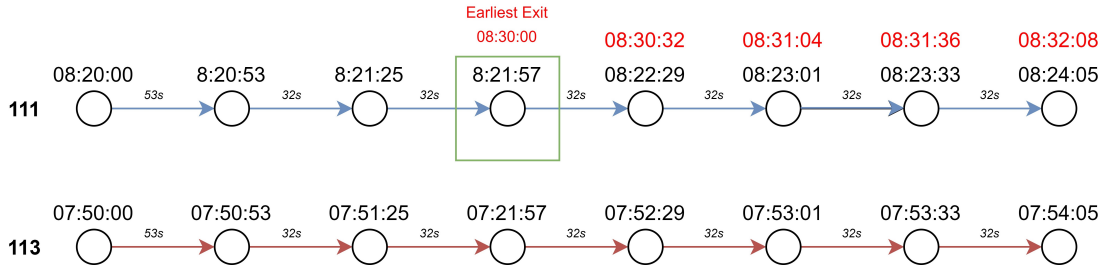
In this section the performance of the algorithm is assessed, using the 9 scenarios as defined at *Train Schedule Optimisation Challenge* 2018. It is known that Scenario 5 cannot be solved with a value of 0 (complying with all hard constraints and no routing penalties). All tests are performed on a series of systems documented in Table 6.5, this was done to calculate the bottleneck of the problem; that is whether it is too intensive on a I/O basis, or more processing power or RAM is what the solver requires. Additionally, operating systems can include



(A) Directed Network Graph



(B) Route Graph Infeasible



(C) Route Graph Forced Feasible

FIGURE 6.2: Sample Scenario Graphs

background software, it would be useful to run the same operating system on the same hardware but unfortunately that is not possible here. It must be noted that the OS named 'WSL' is the Windows Subsystem for Linux; this is not expected to perform well but results will be useful. An attempt was made to parallelise

ID	CPU	Cores	RAM	Storage	OS
01	i7-4750HQ@2.00GHz	4(8)	8GB	SSD	ArchLinux
02	i7-7700K@4.20GHz	4(8)	16GB	SSD	Windows 10
03	i7-7700K@4.20GHz	4(8)	16GB	SSD	WSL

TABLE 6.5: Description of Systems Solving is Performed On

the program using a bash script, allowing for multiple concurrent versions of the program with different random number generation seeds. Each scenario was run 3 times with 4 parallel instances on each system; the best result is given in Table

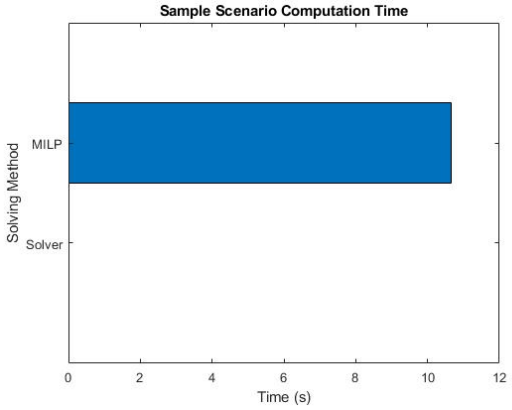


FIGURE 6.3: Sample Scenario Computation Time

6.6 and the associated time to achieve this result. As mentioned in Chapter 4,

	Solving System					
ID	01		02		03	
	$ObjVal$	Time	$ObjVal$	Time	$ObjVal$	Time
01	0	5.452	0	4.296	0	7.590
02	0.43	87	0.43	72.087	0.43	77.042
03	0.86	45	0.96	198.273	0.11	200.234
04	24.94	430	24.94	439.121	024.94	435.945
05	n/a	1800	n/a	1800	n/a	1800
06	207.12	1355	207.12	1174.690	207.12	1750.154
07	456.60	1800	442.33	1800	467.10	1800
08	153.23	1800	122.70	1800	233.6	1800
09	138.95	1800	38.25	1800	43.95	1800

TABLE 6.6: Objective Values for All Scenarios

reinforcement learning techniques require stopping conditions; the stopping condition applied for the results obtained in Table 6.6 was a 1800 second run time (30 Minutes). All simulations of 07/08/09 hit this barrier with a non-optimal solution, scenario 05 was unable to give a solution - this is due to the type of complexity in the scenario, there are line sections blocked which makes it difficult in certain scenarios, of which the Q-Learning algorithm did not know how to deal with it. The only scenario that is solved perfectly is 01, which is incredibly simple and could be done by hand, all other scenarios are sub-optimal however the purpose of this algorithm was not for optimality; it is a proof of concept. It is not feasible for the solver to take an infinite amount of time to get a solution, so the time stopping condition was removed to see if any results could be obtained in longer periods

of time (All these were performed on system 02). The significant results of this were for scenario 09, after running for 19 hours (specifically 70,237 seconds), the objective value was improved to 31.56; not much improvement for the increased computation time. Another significant result was for scenario 07, the simulation only required another 57 seconds to improve the objective value to 374.

Chapter 7

MILP Train Re-Scheduling Problem

7.1 Introduction

Trains are usually schedules in a cyclic format, see figure 7.1. This is to minimise solving complexity for the majority of services, as what works for 1 hour, will usually work for all hours. The other benefit is that it is then easier for passengers to remember timetables as xx:51 (51 minutes past whichever hour). However trains are often delayed, for one reason or another, this leads to knock on delays. The MILP process described in this chapter is often integrated into control systems and can be considered operational scheduling, the MILP will attempt to minimise overall delay and try to get trains as close to timetable as possible. Sometimes an acyclic timetable, see figure 7.2, can perform better for delay recovery due to the larger gaps in timetable providing no blockages.

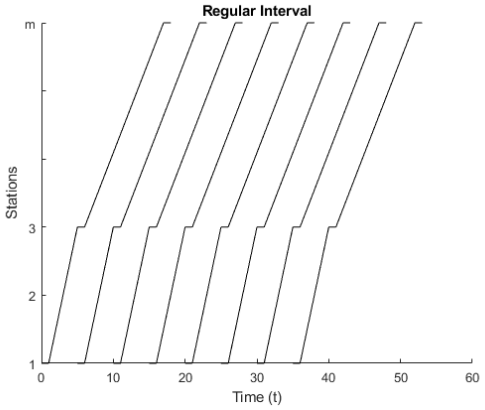


FIGURE 7.1: Cyclic Timetable

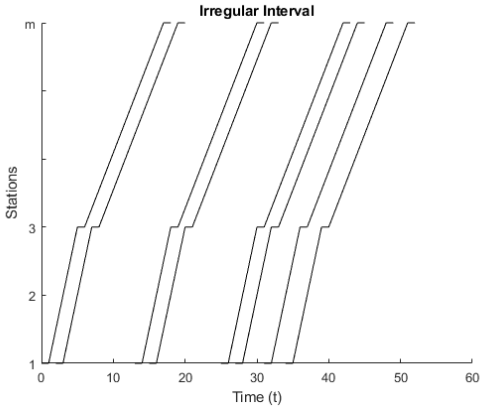


FIGURE 7.2: Non-Cyclic Timetable

7.2 Formulation

7.2.1 Basic TRSP

The Basic Train Re-Scheduling Program is used where a known delay will occur in a short horizon, for example emergency engineering works. This algorithm would be a last resort, if a service still needed to run in harsh conditions. There are n trains moving along a single track with m stations. (In more advanced simulations using this algorithm, a standard dual track is modelled as 2 single tracks in opposite directions.) Until the delay all trains will run on time. The objective is to reschedule the trains to minimise the changes to the timetable. Additionally, trains are not able to overtake preceding trains.

7.2.1.1 Variables

All variables required for the model are defined as:

Variable	Description
N	Set of Trains $n_1, n_2, \ldots, n_N$
M	Set of Stations $m_1, m_2, \ldots, m_M$
T	Maximum Simulation Time
C	Maximum Simulation Capacity
H	Minimum Headway Capacity
$A^o_{n,m}$	Original Scheduled Arrival Time of Train n at Station m
$A^r_{n,m}$	Re-Scheduled Arrival Time of Train n at Station m
$A^r_{n,M}$	Re-Scheduled Arrival Time of Train n at Station M
$D^o_{n,m}$	Original Scheduled Departure Time of Train n at Station m
$D^r_{n,m}$	Re-Scheduled Departure Time of Train n at Station m
$D^r_{n,M}$	Re-Scheduled Departure Time of Train n at Station M
δ_t	Time of Delay Occurring
δ_n	Initially Delayed Train
δ_d	Duration of Delay
δ_D	Destination When Delayed
$\lambda_{n,m}$	Dwell Time for Train n at Station m

TABLE 7.1: Modelled Variables

7.2.1.2 Objective Functions

Objective Function : Minimise Timetable Difference

$$\sum_{n=1}^{N} A^r_{n,M} - A^o_{n,M} \tag{7.1}$$

7.2.1.3 Constraints

The first constraint modelled is Equation (7.2). It states that all Trains originate at $t = 0$:

$$A^r_{n,1} = 0 \tag{7.2}$$

All Trains depart last station when they arrive:

$$D^r_{n,m} = A^r_{n,m} \tag{7.3}$$

Before the delay is introduced all trains should run on schedule (formally if $D^r_{n,m} < \delta_t$):

$$D^r_{n,m} = D^0_{n,m} \tag{7.4}$$

If train is moving and will become delayed at some point if $\delta_D > 1$ and $\delta_t < D^{\delta_D - 1}_{\delta_n} + E_{\delta_D - 1}$ then:

$$A^r_{\delta_n,\delta_D} \geq D^0_{\delta_n,\delta_D - 1} + \delta_d + E_{\delta_d - 1} \tag{7.5}$$

Delayed Trains next departure must incorporate delay:

$$D^r_{\delta_n,\delta_D} \geq \delta_t + \delta_d \tag{7.6}$$

Trains must depart after they have arrived:

$$D^r_{n,m} \geq A^r_{n,m} \tag{7.7}$$

Trains must depart after they have arrived:

$$D^r_{n,m} \geq A^r_{n,m} \tag{7.8}$$

Trains cannot depart earlier than the original schedules departure:

$$D^r_{n,m} \geq D^o_{n,m} \tag{7.9}$$

Trains cannot arrive at the next station faster than the minimum distance time:

$$A^r_{n,m+1} \geq D^r_{n,m} + E_m \tag{7.10}$$

Trains must leave in their order $n_0 \to n_1 \to \cdots \to n_N$:

$$D^r_{n,m} > D^o_{n+1,m} \tag{7.11}$$

More than one train cannot occupy the same station (however, they can skip the station (perhaps an express line through the station)):

$$D^r_{n,m} \leq A^r_{n+1,m} - H \tag{7.12}$$

Time dwelled at a station is modelled as:

$$\lambda_{n,m} = D^r_{n,m} - A^r_{n,m} \tag{7.13}$$

7.2.2 Constant Passenger Demand TRSP

The addition of passenger demand adds significant complexity to the model, which increases computation time. All passengers destinations are assumed to be the final station (similar to a commuter train heading towards either Manchester or Liverpool). Additionally, more passengers require longer to board the train and train capacity must be abided by.

7.2.2.1 Variables

The additional variables required to model passenger demand:

Variable	Description
P^b_m	Passenger Initial at Station m
P^s_m	Passenger Final at Station m
P^f_m	Passenger Flow at Station m
$P^c_{n,m}$	Passengers Collected on Train n at Station m
$P^t_{n,m}$	Total Passengers on Train n at Station m
$\sigma^l_{n,m}$	Lower Time Bound on Train n at Station m
$\sigma^u_{n,m}$	Upper Time Bound on Train n at Station m

TABLE 7.2: New Modelled Variables

7.2.2.2 Objective Functions

The objective function is changed to reflect passenger weighting, so instead of just minimising timetable change, we also want to minimise changes to very busy trains

Objective Function : Passenger Weighting

$$\sum_{n=1}^{N} P^{c}_{n,M} A^{r}_{n,M} \tag{7.14}$$

7.2.2.3 Constraints

The sigma values partition time at each station:

$$\sigma^{l}_{n,m} \leq \sigma^{u}_{n,m} \tag{7.15}$$

Extreme Time Bounds:

$$\sigma^{l}_{n,m} = P^{s}_{m} \tag{7.16}$$

$$\sigma^{u}_{N,m} = D^{o}_{n,m} \tag{7.17}$$

Time Windows are joined together at each end:

$$\sigma^{u}_{n,m} = \sigma^{l}_{n+1,m} \tag{7.18}$$

Each σ value is defined, there are two seperate for n and N:

$$\sigma^{u}_{n,m} = D^{r}_{n,m} \tag{7.19}$$

$$\sigma^{u}_{N,m} = D^{r}_{N,m} \tag{7.20}$$

Passengers that will be picked up:

$$P = P^{f}_{m}\left(\sigma^{u}_{n,m} - \sigma^{l}_{n,m}\right) \tag{7.21}$$

$$\sigma^{l}_{n,m} = P^{c}_{n,1} \tag{7.22}$$

$$\sigma^{l}_{n,m} = P^{c}_{n,m-1} + P^{c}_{n,m} \tag{7.23}$$

$$\tag{7.24}$$

Boarding Time Calculations, if there are less than 100 passengers it takes $20t$, if there are more, it takes $50t$:

$$C - P^{l}_{n,m-1} < 100 \rightarrow P^{n,m}_{c} \leq 20\lambda_{n,m} \tag{7.25}$$

$$C - P^{l}_{n,m-1} \geq 100 \rightarrow P^{n,m}_{c} \leq 50\lambda_{n,m} \tag{7.26}$$

$$P^{c}_{n,m} \leq D + n, m^{r} - 50A^{r}_{n,m} \tag{7.27}$$

7.3 Computational Model

7.3.1 MiniZinc

This basic model will utilise flattened MinZinc to solve the problem as defined in this section. On MiniZincs website they state that MiniZinc is a medium-level constraint modelling language. It is high-level enough to express most constraint problems easily, but low-level enough that it can be mapped onto existing solvers easily and consistently. It is a subset of the higher-level language Zinc. Additionally the developers hope it will be adopted as a standard by the Constraint Programming community. (*MiniZinc* 2020). FlatZinc is a low-level solver input language that is the target language for MiniZinc. It is designed to be easy to translate into the form required by a solver. Constraint solvers do not support MiniZinc models natively, so to run a MiniZinc model it is translated into FlatZinc. Most constraint solvers only solve satisfaction problems of the form $exists(c_1 \wedge \cdots \wedge c_m)$ or optimisation problems of the form $min\ z$ s.t. $c_1 \wedge \cdots \wedge c_m)$ where c_i are primitive constraints and z is an integer or float. (A FlatZinc model is hardcoded with a data set, and must be generated for each variation.) A main reason that MiniZinc was chosen above all other solver solutions was due to it being a higher-level language than solvers, and as such can be imported into each solver, the main solvers that will be discussed in this paper will be (*Gurobi* 2020) and (*Google OR-Tools* 2020). Solver Performance will also be investigated for varying data sets.

7.3.1.1 Constraints

The constraints defined in Section 7.2.1.3 and 7.2.2.3 are Coded in MiniZinc. Each constraint is detailed here:

For this model a series of constraints are required and defined as follows: Each constraint is marked with the identifier *constraint* and have a comment attached.

```
% All trains "arrive" at the first station at time 0.
constraint forall (i in 1..n)
        (arrival[i,1] = 0);
% ... and "depart" from the last station as soon as they
  arrive there.
constraint forall (i in 1..n)
        (departure[i,m] = arrival[i,m]);

% Before the delay, everything runs to schedule.
constraint forall (i in 1..n, j in 1..m-1)
        (if scheduledDeparture[i,j] <= delayTime
           then departure[i,j] = scheduledDeparture[i,j]
           else true
         endif);
```

Listing 7.1: TRSP Constraints 1-4 Code Snippet

```
constraint if destinationWhenDelayed > 1
           then if delayTime <
 scheduledDeparture[delayTrain,destinationWhenDelayed-1]
 + distance[destinationWhenDelayed-1]
               then
 arrival[delayTrain,destinationWhenDelayed] >=

 departure[delayTrain,destinationWhenDelayed-1] +
 delayDuration + distance[destinationWhenDelayed-1]
               else true
```

```
6                endif
7            else true
8            endif;
9 % The train's next departure is at least the delay time
   plus the delay
10 % duration.
11 constraint departure[delayTrain, destinationWhenDelayed]
   >= delayTime + delayDuration;
```

Listing 7.2: TRSP Constraints 1- 4 Code Snippet

```
1 constraint forall (i in 1..n, j in 1..m)
2         (departure[i,j] >= arrival[i,j]);
3
4 % Trains never leave earlier than scheduled.
5 constraint forall (i in 1..n, j in 1..m-1)
6         (departure[i,j] >= scheduledDeparture[i,j]);
7
8 % There is a minimum travel time between stations.
9 constraint forall (i in 1..n, j in 1..m-1)
10         (arrival[i,j+1] >= departure[i,j] + distance[j]);
11
12 % At station 1, trains leave in order.
13 constraint forall (i in 1..n-1)
14         (departure[i,1] < departure[i+1,1]);
15 % At most one train dwelling at a station at a given
   time.
16 constraint forall (i in 1..n-1, j in 2..m-1)
17         (departure[i,j] <= arrival[i+1,j]-2);
```

Listing 7.3: TRSP Constraints 5- 8 Code Snippet

```
1 constraint forall (i in 1..n, j in 1..m)
2         (sigmaLower[i,j] <= sigmaUpper[i,j]);
```

```
3
4 % For the first and last trains, the sigma values are
   equal to the
5 % extreme times of passenger arrivals.
6 constraint forall (j in 2..m-1)
7         ((sigmaLower[1,j] = passengerStart[j])
8      /\ (sigmaUpper[n,j] = scheduledDeparture[n,j]));
9 % The sigma values join together.
10 constraint forall (i in 1..n-1, j in 1..m)
11         (sigmaUpper[i,j] = sigmaLower[i+1,j]);
12
13 % You can't pick up people after you leave.
14 constraint forall (i in 1..n-1, j in 1..m-1)
15          (sigmaUpper[i,j] <= departure[i,j]);
16 constraint forall (j in 1..m-1)
17         (sigmaUpper[n,j] <= departure[n,j]);
```

Listing 7.4: TRSP Constraints 5- 8 Code Snippet

```
1 constraint forall (i in 1..n, j in 1..m)
2         (collect[i,j] =
    (sigmaUpper[i,j]-sigmaLower[i,j])*passengerFlow[j]);
3
4 constraint forall (i in 1..n)
5         (load[i,1] = collect[i,1]);
6 constraint forall (i in 1..n, j in 2..m)
7         (load[i,j] = load[i,j-1] + collect[i,j]);
8
9 % If a train picks anyone up, then it must pick
10 % everyone up (until it gets full).
11 constraint forall (i in 1..n, j in 1..m-1)
12         (sigmaUpper[i,j] > sigmaLower[i,j] ->
13              ((sigmaUpper[i,j] = departure[i,j]) \/
```

```
14              (sigmaUpper[i,j] =
  scheduledDeparture[n,j]) \/
15              (load[i,j] + bool2int(sigmaUpper[i,j] <
  scheduledDeparture[n,j])*passengerFlow[j] > capacity)));
16
17 % Boarding time.
18 constraint forall (i in 1..n, j in 2..m)
19         (((capacity-load[i,j-1] < 100) -> (collect[i,j]
  <= dwell[i,j]*20)) /\
20          ((capacity-load[i,j-1] >= 100) -> (collect[i,j]
  <= dwell[i,j]*50)));
21 % (Redundant for j>=2 (but necessary for j=1))
22 constraint forall (i in 1..n, j in 1..m)
23         (collect[i,j] <=
  (departure[i,j]-arrival[i,j])*50);
24
25
26 constraint forall (i in 1..n, j in 1..m)
27         (dwell[i,j] = departure[i,j] - arrival[i,j]);
```

Listing 7.5: TRSP Constraints 5- 8 Code Snippet

7.3.1.2 Objective Function

The objective function is defined as a range of values on the optimal plane: that being the plane *objective_min* → *objective_max*. It is performed in this method because of the searching algorithm that must be used to find the solution defined in Section 7.3.1.3

```
1 int: objective_min = lb(sum(i in
  1..n)(load[i,m]*arrival[i,m]));
2 int: objective_max = ub(sum(i in
  1..n)(load[i,m]*arrival[i,m]));
```

```
3 var objective_min..objective_max: objective = sum(i in
   1..n)(load[i,m]*arrival[i,m]);
```

Listing 7.6: TRSP Objective Function Code Snippet

7.3.1.3 Solving Method

The solving method used by the algorithm is a branch and bound searching algorithm, in MiniZinc this is defined as in Listing 7.7. The general idea is shown in Figure 7.3.

```
1  solve
2      :: seq_search([
3          int_search(
4              [arrival[i,m-j+1] | j in 1..m, i in 1..n] ++
5              [departure[i,m-j+1] | j in 1..m, i in 1..n]
   ++
6              [sigmaUpper[i,m-j+1] | j in 1..m, i in 1..n]
   ++
7              [sigmaLower[i,m-j+1] | j in 1..m, i in 1..n],
8              input_order, indomain_min, complete
9          ),
10         int_search(
11             array1d(1..n*m, collect) ++ array1d(1..n*m,
   load) ++ array1d(1..n*m, dwell),
12             first_fail, indomain_min, complete
13         )
14     ])
15     minimize objective;
```

Listing 7.7: TRSP Solving Search Function Code Snippet

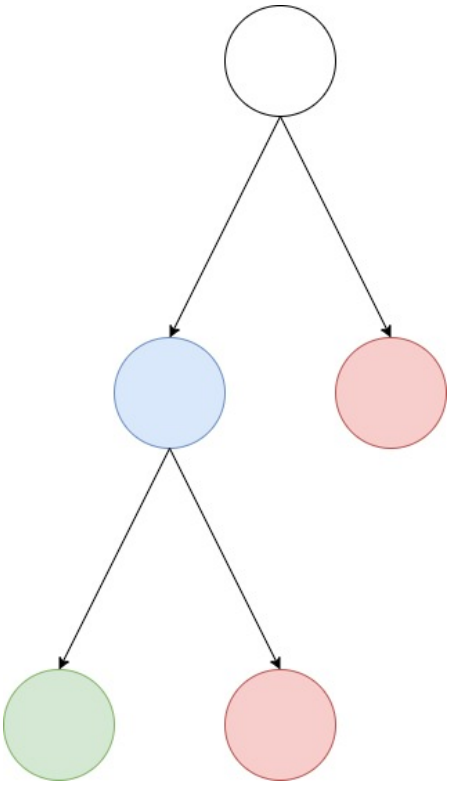


FIGURE 7.3: Example of the Branch and Bound Method

7.3.2 FlatZinc

FlatZinc is a low level language, it is what the high level language MiniZinc is translated into, so that it can be understood by solvers. When solving a MiniZinc problem this is done by interpretation (line by line style) which negatively affects performance but is good for debugging. There is an option to compile into FlatZinc which is great for solving larger problems, but cannot be understood easily (the problem defined above is roughly 16,000 lines in FlatZinc, compared to 180 in MiniZinc). FlatZinc has direct interfaces to COIN-OR CBC, Gurobi and CPLEX. Additionally, OR-Tools developed a FlatZinc interface themselves. A FlatZinc model consists of the following:

1. external predicate declarations,
2. parameter declarations,
3. variable declarations,
4. constraints,

5. a solve goal,

in this specific order.

7.3.3 Solvers

The solvers that this project is interested in are Gurobi and Google OR-Tools; Gurobi is one of the fastest growing solvers currently, and is very widely used for commercial interests. It is a very high performance solving framework and is designed for ease of use. Google OR-Tools is high performance, but is open source, OR-Tools performs the best of open source solvers and compares to Gurobi for some solving methods however Gurobi is far better suited for this project.

7.4 Results

Both solvers will receive the same compiled FlatZinc problem with the same data set; the objective value and computation time will be evaluated and compared. Since there are so many data sets the results are divided into categories; Small, Medium and Large, these represent the size of the data set. Although the problem is extremely simplistic compared to a real life problem, solving time is extremely long for any problem with more than 3 trains due to the massive knock on effects that occur. (The original plan for the results was to demonstrate solver performance on each problem, ranging from 1 train to 200 trains. However the 5 train problem takes 400 seconds to solve, and an attempt at solving 100 trains was made but never finished due to the computation time required.)

7.4.1 Small Problem

The smallest set of data is $n = 3$. That being 3 trains, all scenarios have 5 stations unless otherwise stated. The delay occurs on the second train at minute 5 with a duration of 5. The problem solved in MiniZinc, requiring live interpretation, has a solve time of 1037ms. If compiled into a fzn file solving takes 80ms, and OR-Tools solves in 96ms. The objective value attained by all solvers is 34020 requiring 703 nodes.

7.4.2 Medium Problem

The medium problem is $n = 5$, all static information is the same as in the small problem. Solving for this problem takes 407.6534 seconds using fzn. Achieving an objective value of 71820 using 486811 nodes. The mzn file solves in 576s and OR-Tools solves in 432.43s.

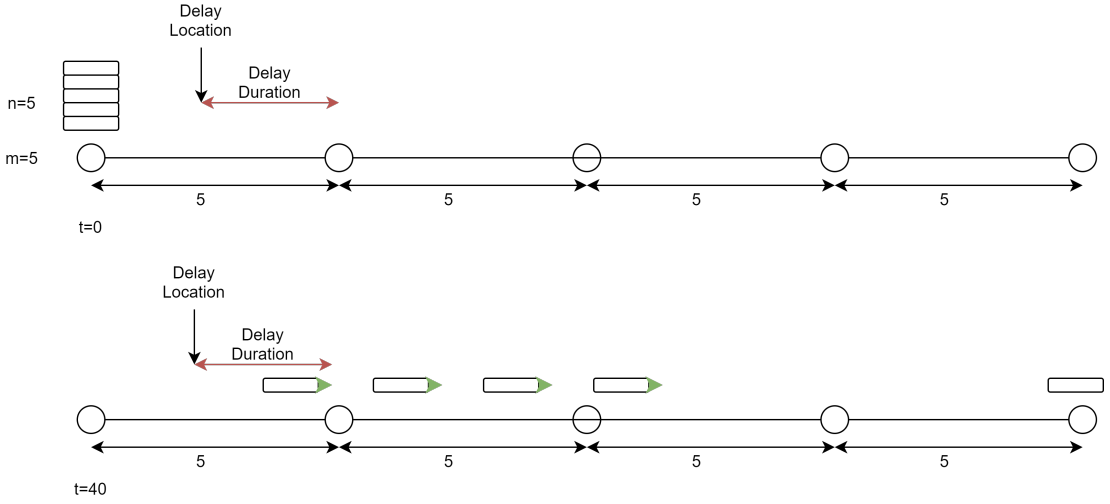


FIGURE 7.4: Dual State of Medium MILP Route Graph

7.4.3 Large Problem

The large problem consists of two variants of a 10 train problem, the variants are 5 stations and 10 stations. This could accurately simulate a 2 hour window on a single line. This problem was given 8 hours to solve, and was unfeasible from both solvers. This demonstrates the inherent problem with ILP Branch and Bound solving method where the number of nodes becomes far too large to find a solution in a realistic amount of time. This is a very simple problem compared to that discussed in Chapter 6. A demonstration of how the system looks when solved manually is demonstrated in Figures 7.5 at $t = 40$ 7.6 for $m = 5$, and for $m = 10$ in 7.7 and at $t = 40$ in 7.8.

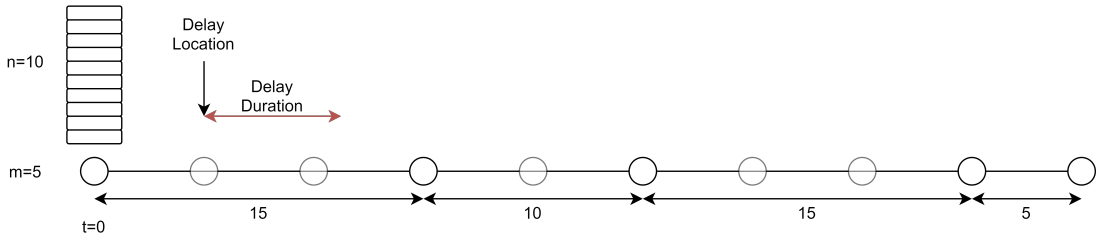


FIGURE 7.5: $m = 5$ Initial State of Large MILP Route Graph

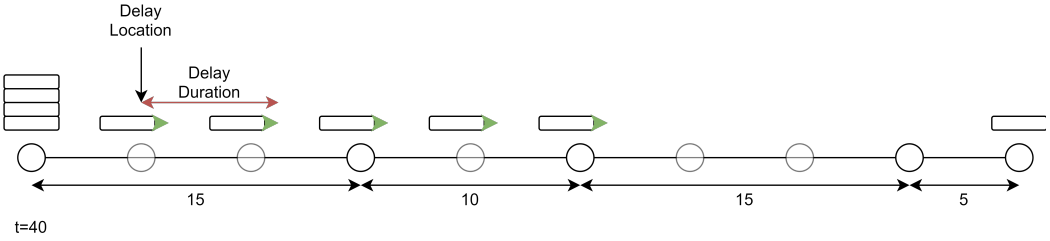


FIGURE 7.6: $m = 5$ State of Large MILP at $t = 40$ Route Graph

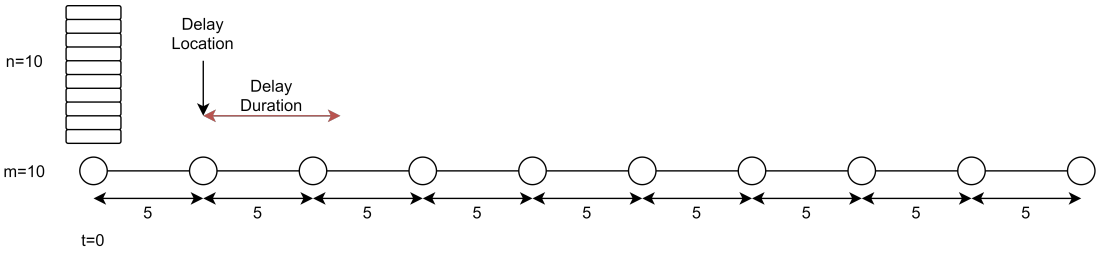


FIGURE 7.7: $m = 10$ Initial State of Large MILP Route Graph

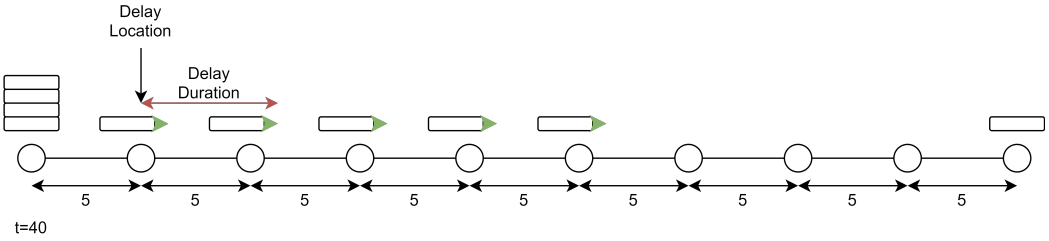


FIGURE 7.8: $m = 10$ State of Large MILP at $t = 40$ Route Graph

7.5 Discussion

The basic TRSP presented in this chapter provides a clear reason for the avoidance of ILP for complex problems, even though they can be useful, they are largely outperformed by other algorithm methods such as Heuristics or ML. Before running the problem scenarios, the author designed 15 different problem scenarios up to

$n = 50, m = 50$ and $n = 200$. This was revised once the $n = 10$ problem could not be solved in 8 hours.

Chapter 8

Evaluating Manchester Corridor

8.1 Introduction

In the May 2018 timetable revision, the Trains Per Hour (tph) was increased from 12 to 15 across the Manchester Corridor and it brought massive delays. The corridor is not able to cope in live scenarios. The Manchester Corridor is a significant pinch point for East-West operations, as well as all services operating through Manchester Piccadilly from the West. Although there are plans in place to increase overall capacity to 16tph, this chapter will investigate whether it is possible, and if the current infrastructure is not capable. Suggesting changes to increase capacity and robustness.

8.1.1 Assumptions

The infrastructure model used in this are based on May 2019 timetable, geographic data attained through Chapter 3. Additionally the analysis on junction utilisation is based on May 2019 timetable, where a period of 7 days were recorded (August 11 - 18 2019) and all trains moving across each junction over a rolling hour period are averaged. Train lengths are assumed to be a variable length, where all lengths are in Appendix C.1.

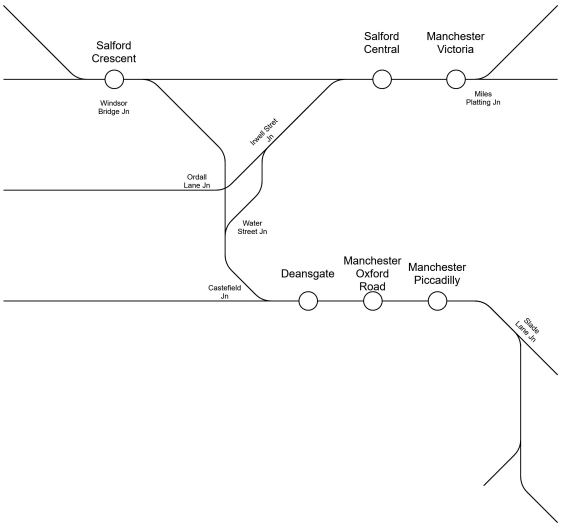


FIGURE 8.1: Manchester Stations Track Map

8.1.2 Methodology

The analysis in the following section was developed based on data attained from the system created in Chapter 3, additionally the author spoke to operational personnel from East Midland Regional and Grand Central; whereby a discussion about scheduling procedures occurred.

8.2 Analysis

8.2.1 Track Capacity

It has been identified through speaking to operational personnel at TOC's that some Timetable Planning Rules are not feasible for real-life usage. This can be seen in the timing data from Chapter 3, that platform reoccupation on Manchester corridor is a constraint. The current TPR's specify a headway of 2 minutes (*Network Rail Operational Rules* 2020), the problem occurs when the length of the train is too great to fit in a single signal zone (like most units on the route do) and leaves no time margin for error or even passive train driving styles. As train lengths increase, the headway time must increase too and this would cause the current limitation of 15 TPH on the Manchester Corridor to be impossible without significant infrastructure changes.

8.2.2 Manchester Oxford Road

Manchester Oxford Road has 4 through platforms and 1 bay platform, although for most services only platform 2 and 4 are used. Due to the layout of MCO's platforms and signal sections, any train longer than 80m cannot arrive or depart simultaneously with another. All trains that pass through Manchester Corrridor can be seen in Appendix C.1. Often operators will run train sets combined which would double the length stated in the table, however most trains do qualify to be able to arrive/depart simultaneously. Train Planning Rules state that a 2-minute dwell is required for all services at MCO, in addition to the 2-minute platform reoccupation, therefore Oxford Road can platform a train every 4 minutes in each direction. Indicative Train Service Specification currently requires 13-TPH to use the through platforms, this gives platforms 2 and 4 a utilisation rate of 87% which is extremely high compared to all stations (including London Terminus). There are also 2 TPH that use the bay platform (Platform 5), since they do not use platform 2 or 4, the re-occupation is not neccessary, only headway constraints. Giving Oxford Road a total utilisation rate of 93% (56/60 minutes) every hour, not including freight trains which require to pass through the station frequently from Trafford Park. It should be noted that the 2-minute Dwell Time is highly unachievable based on local observation by the author, reasons that this is not achievable include extremely busy trains, other blockages or crew changeovers. Double movements as described in Figure 8.2 would allow Manchester Corridor Timetables to either:

1. Increase TPH
2. Increase Timetable Robustness

Figure 8.2 demonstrates the proposed form of double movements, the double movements are only ever performed in a single direction, it is always platform 3 that is the secondary platform due to Platform 1 not having step free access making it infeasible. In this model there are 3 trains; Green, Blue and Red. The primary platform is Platform 4, since it is empty when Green arrives, green will move to Platform 4 and deloads passengers. As Blue arrives in the model, the

Primary platform is occupied, so will assume the secondary platform, Platform 3 (usually it would stay at Deansgate Station, waiting for the Primary platform to clear (2 minute minimum)). When the Blue train starts to deload, Green has finished and the exit is clear, so exits the model. Red enters the model and since the primary platform is empty, goes to Platform 4, as Blue leaves. And so on. Assuming that all trains are below 80m, this could increase Manchester Oxford Road's theoretical maximum TPH to 30-TPH, however this ignores all stations either side, which would not be able to cope. At the current 13-TPH, with the double movement model suggested here, platform 4 utilisation would have halved, with average platform utilisation increasing. Since not all trains are below 80m long, new rolling stock timetabling constraints could be introduced to plan lengths of train into the schedule to allow for pre-planning of station allocations, it could also be done on a short-term basis, as TOC's know what train unit and therefore length of unit will be operating which services. This would allow Network Rail to allocate platforms on the day and timetable assuming that not all trains will be able to perform it. The third potential plan would be to implement it without any timetable changes, and doing it as and when possible to increase robustness of timetables and helping mitigate delays. Increasing robustness of timetables is advantageous for timetabling planning as complex junctions are scheduled with little down time, so increasing on-time presentation at key junctions in Manchester such as Water Street Junction.

8.2.3 Service Limitations

As noted in the previous section, there are currently too many trains scheduled onto the Manchester Corridor to be operated robustly. The solution is either infrastructure based or timetabling based as described in the previous section. Infrastructure must be changed to allow for more TPH, changes are suggested in section 8.2.4 A timetabling based change not discussed in the previous section would be running a reduced service to allow for some services to have greater margin for error and also reducing track utilisation to around 85% which is the average suburban rail network utilisation rate. There are several services identified

as problematic due to the number of junction interactions where other trains must stop.

8.2.3.1 Manchester Aiport to Middelesbrough Service

A train heading from Manchester Airport towards Middlesbrough requires conflicting moves at Castlefield Junction, Water Street Junction, Irwell Street Junction (Figure 8.3) and then (depending on platform allocation at Victoria, or junction capacity at Deal Street Junction) Victoria East Junction or Miles Platting Junction. This makes it a very difficult train to plan as, due to the number of problematic moves, it is the least flexible of all services running through Manchester and therefore other services must be planned around it. Any other train planned in or out of Victoria, or in to Oxford Road, would conflict with this service. The only complete non-conflicting movement that can take place whilst the train is passing through the Corridor is to have a train running the other way along the Ordsall Chord, from Manchester Victoria to Manchester Oxford Road. This service could be made marginally better by using Platform 4 instead of Platform 6 at Manchester Victoria which would move one of the conflict points, by crossing services to the Calder Valley line at Miles Platting Jn or immediately to the east of Victoria station.

8.2.3.2 Southport to Alderley Edge Service

A train heading from Southport towards Alderley Edge requires conflicting moves at Windor Bridge South Junction, Ordsall Lane Junction and Castlefield Junction (Figure 8.4). This is a typical service coming into Manchester Oxford Road, and requires multiple blocked routes as highlighted by the arrows in Figure 8.4

8.2.3.3 Liverpool to Manchester Victoria Services

A train heading from Liverpool towards Manchester Victoria along the Northern route requires conflicting moves at Ordsall Lane Junction and Irwell Street Junction (Figure 8.5). Most services towards Liverpool via Oxford Road use the Southern route, highlighted in blue and purple, which helps alleviate pressure on

Ordsall Lane Junction, and as can be seen in Figure 8.5 there are only junction blocking movements in a single direction. Services that go via Manchester Victoria use the Northern route, light green and red, which are capable of running parallel but do contain junction blocking movements and can cause congestion. It can be seen, in Table 8.1, that due to this timetabling decision, Castlefield Junction has the highest number of movements per hour whereas Ordsall Lane has one of the fewest because of all the blocking that a train does whilst crossing. For maximum movements per hour, less complex junctions are better due to fewer blocking scenarios.

8.2.4 Infrastructure Limitations

8.2.4.1 High Junction Utilisation Rate

Data analysed from Chapter 3 suggests that the junctions in Manchester are saturated. A high number of trains cross key junctions, for example, 30 trains cross Castlefield Junction (Figure 8.1). The high number of trains leaves very little time for recovery space or additional freight trains. Without a complete infrastructure overhaul nothing can be done about these junction utilisation rates.

Junction	Movements
Castlefield	30
Windsor Bridge North	22
Windsor Bridge South	22
Water Street	20
Ordsall Lane	16
Irwell Street	12

TABLE 8.1: Train Movements of Trains at Junctions in Manchester

8.2.4.2 Manchester Corridor

The capacity of Manchester Corridor is pushed to the absolute limit, and this is reflected in repeated delays and cancellations. It was identified in Section 8.2.1 that using the additional platforms at Oxford Road would be beneficial, in extension to this it would be massively beneficial to have additional platforms at Manchester Piccadilly or a 3rd line from Oxford Road to Picadilly: the cost of this would be

huge but the impact would be huge and would make the timetable very robust for all scenarios. This would operate the same as suggested in Figure 8.2, where the current eastbound track would be converted into a uni-directional track based on which direction requires more capacity at the time: similar to escalators in London Underground Stations.

8.2.4.3 Salford Crescent

Salford Crescent has 22 TPH, Windsor Bridge North and Windsor Bridge South on either side of Salford Crescent, in Figure 8.7 the red arrows show potential collisions. All trains from every direction must make at least one conflicting move, except for trains from Bolton towards Victoria. A potential solution would be a 3rd Platform on the additional line, this would negatively affect TransPennine Express services towards Scotland however it would benefit all other services. The same middle track configuration would apply here as described previously. With the exception that when operating eastbound trains towards Victoria would be prioritised to the far left platform (the new platform) and services towards Piccadilly would be prioritised to the middle platform (current Platform 1). Operating Westbound, the far right platform (current Platform 2) would be prioritised for services towards Atherton, and middle platform for services towards Bolton. This mitigates some conflicting moves, but they largely remain, this change would be for overall capacity and would require further analysis for how to extract the most 'value'.

8.2.4.4 Ordsall Lane Jn

A grade separation at Ordsall Lane Jn would increase capacity of east-weight north-south traffic and would decrease reliance on parallel movements however would create bottlenecks in other places. This would be prohibitively expensive and would cause delay for multiple years. Alternatively the schedule could be created with the focus on trains arriving at key junctions at a time for complete parallelisation, this would get affected by late running trains, perhaps reducing quantity of trains for longer trains would be beneficial. Trains that run through

both Manchester Victoria and Piccadilly could instead terminate at Victoria and remove pressure on Water Street Junction and Castlefield Junction. The lines in blue are the current configuration and the red lines are the suggested changes.

8.2.4.5 Irwell Street Jn

Irwell Street Junction provides problems for trains operating from East to West, West to East or going between Manchester Stations. Often trains are unable to run in parallel due to both tracks from Ordsall chord filtering into a single track, or forcing trains across aggressively. A change could be to elongate the southbound track of the Ordsall Chord at the north end to be able to run 4 parallel trains, whilst this scenario is not probable it could reduce headway and therefore waiting at berths. This is shown in figure 8.9 in the red track, it is only a track addition.

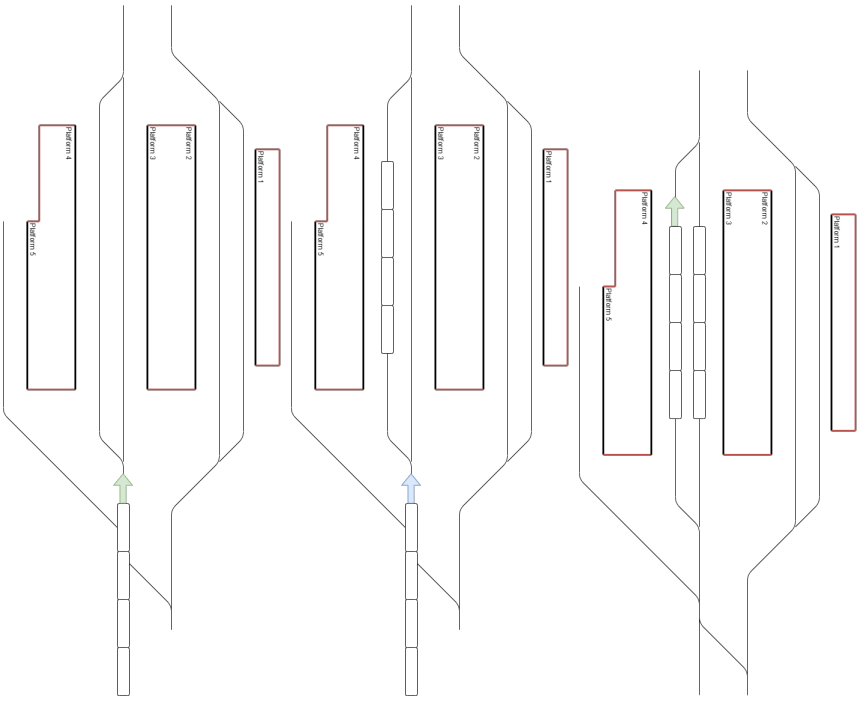
8.2.4.6 Train Lengthening

The use of longer trains in the future prevents the use of any platform at Manchester Airport or Manchester Picadilly by two trains simultaneously. This greatly reduces the capacity of the station and forces departures that may be sub-optimal for utilising capacity in central Manchester in order to ensure platform availability for arriving services. This also drives short turnround times that will likely have a negative impact upon performance network performance or train unit allocations.

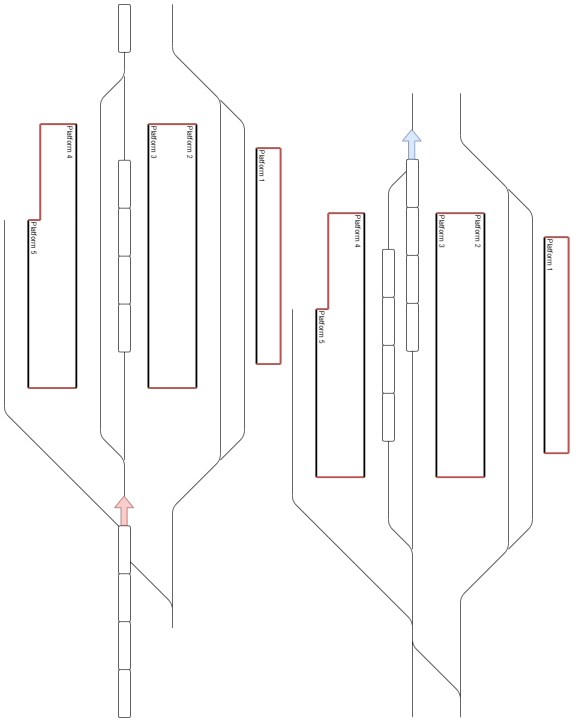
8.3 Conclusion

This chapter has shown that the current timetable is not feasible without some delays, although with the infrastructure changes proposed in Figure 8.10 the theoretical maximum tph is 21tph operating through Deansgate station in each direction; 41 junction movement at Castefield Junction however this would require more thorough testing and a timetable this dense would be highly volatile to delays. The increase in movement to the east of Victoria would allow higher capacity into the station, however Victoria itself has capacity to spare, the routes to get to it are saturated. The proposed changes to Irwell Street Junction would allow trains

to go from the Ordsall Chord without having to stop, additionally the changes to Ordsall Junction (implementing a grade separation) would allow trains from Liverpool to Victoria to not have to stop at any point whilst entering Manchester. Compared to potentially 3 conflicting junction crossings.



(A) Stage 1 (B) Stage 2 (C) Stage 3



(D) Stage 4 (E) Stage 5

FIGURE 8.2: 2 Train Configuration

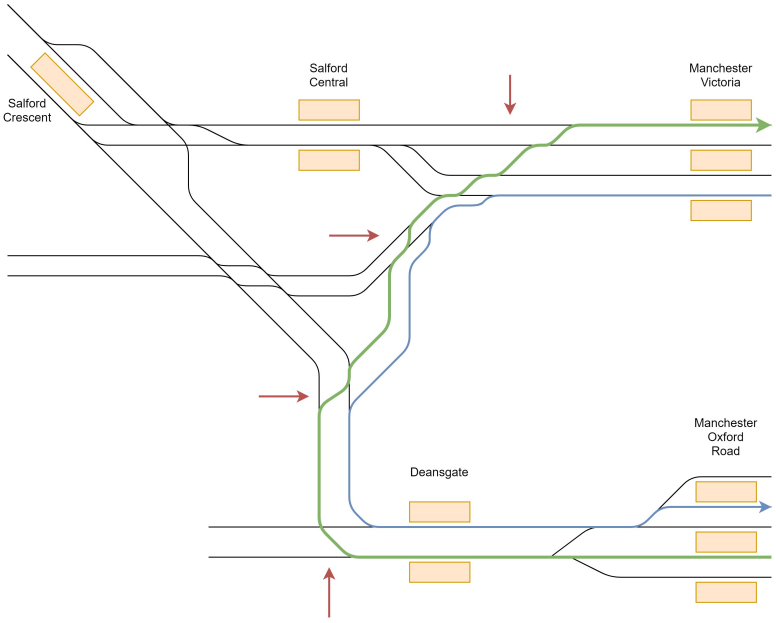


FIGURE 8.3: Junction Map of Manchester Airport to Manchester Victoria

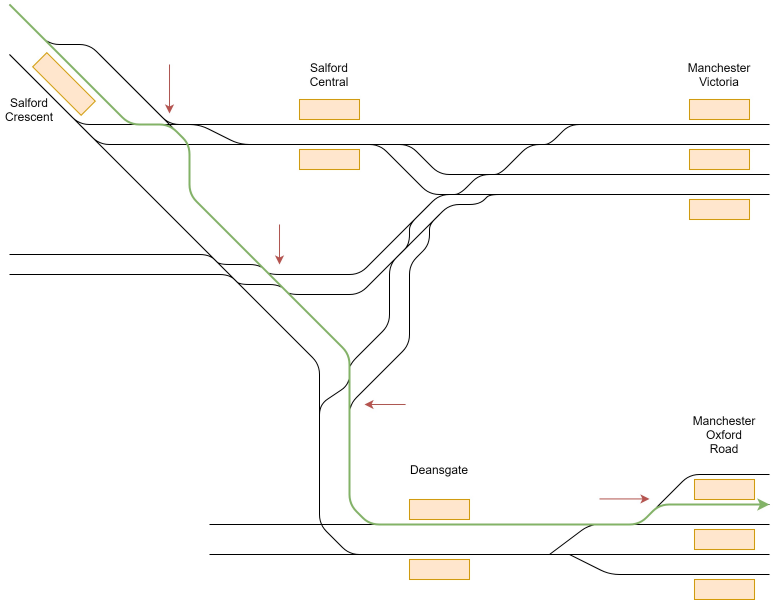


FIGURE 8.4: Junction Map of Southport to Alderley Edge

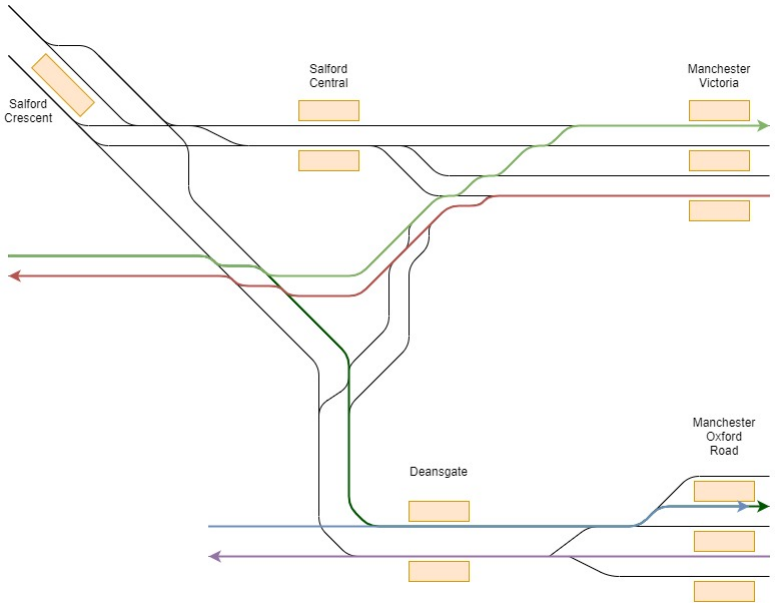
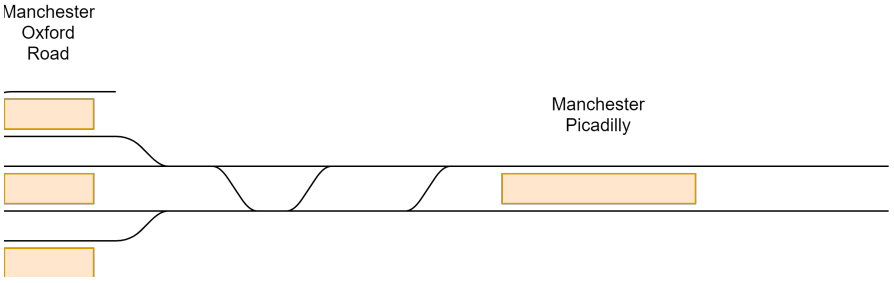
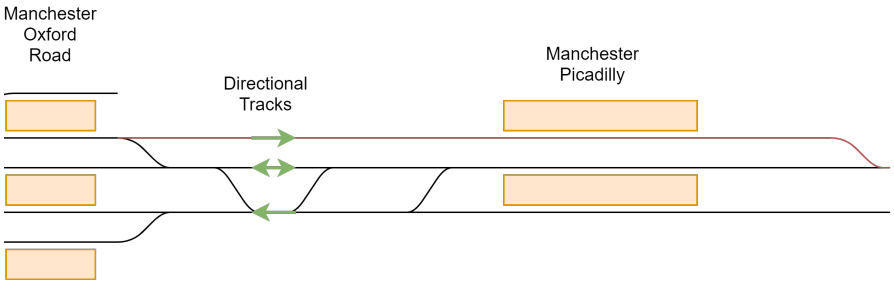


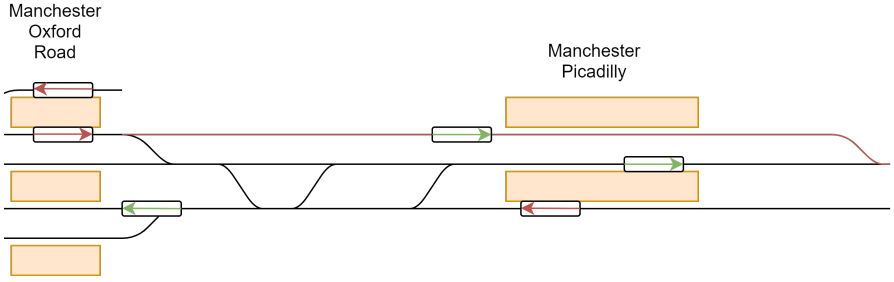
FIGURE 8.5: Junction Map of Liverpool to Manchester Stations



(A) Current Layout



(B) Suggested Changes



(C) Operational

FIGURE 8.6: Junction Map of Manchester Piccadilly

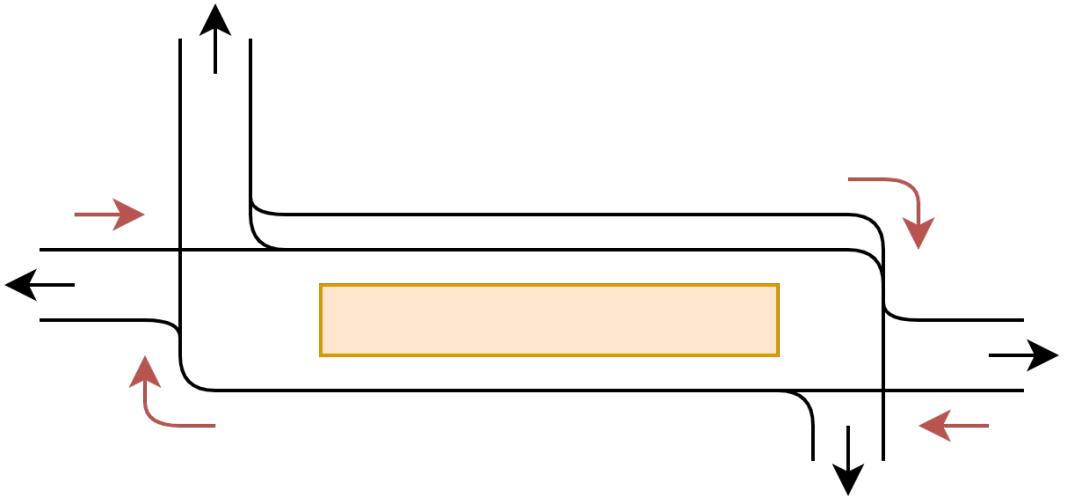


FIGURE 8.7: Salford Crescent Junction Collisions

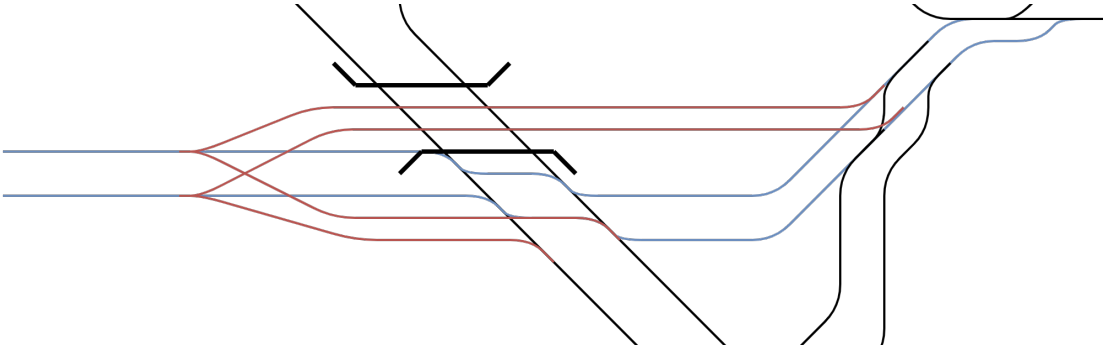


FIGURE 8.8: Ordsall Lane Junction Infrastructure Changes

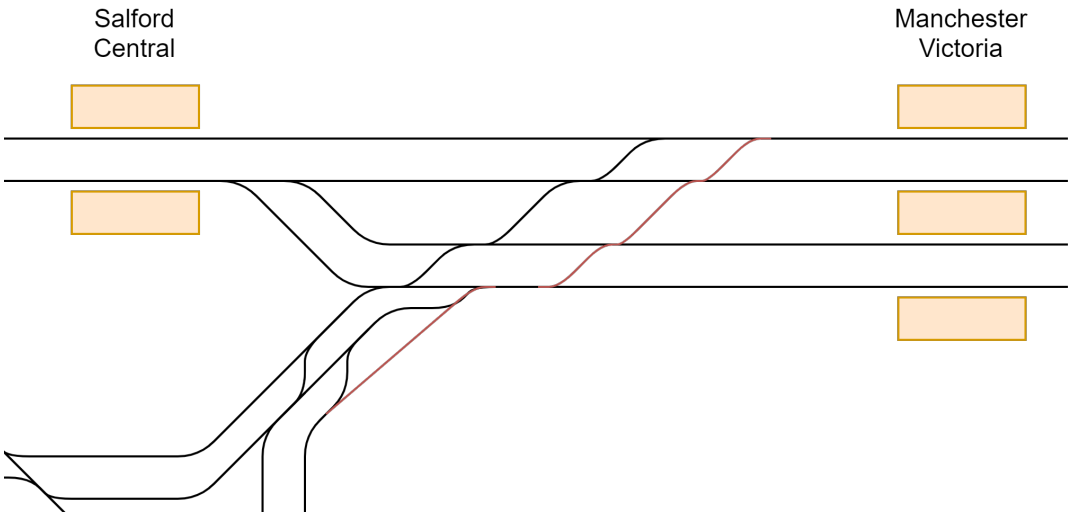


FIGURE 8.9: Irwell Street Junction Infrastructure Changes

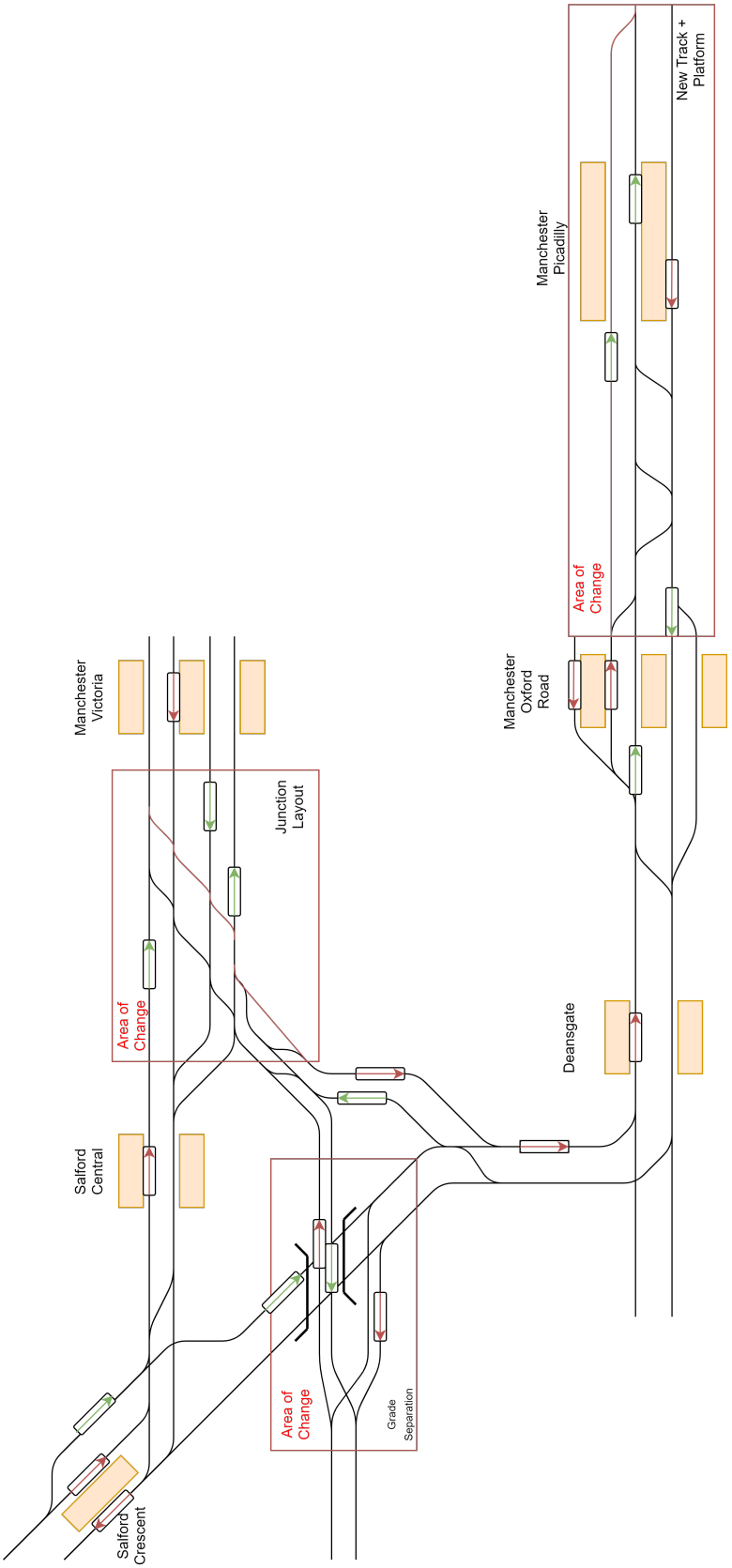


FIGURE 8.10: Manchester Junctions Infrastructure Changes

Chapter 9

Discussion & Future Work

In this chapter, the study on artificial intelligence for the train scheduling problem is concluded. The chapter consists of a short recap over the main research areas, discussion on those, conclusions to the questions from the introduction, and suggestions on future work.

9.1 Research Recap

This research studied the feasibility of using the Genetic Algorithms and Reinforcement Learning in the train scheduling problem. The problem sets may seem segmented, this is due to the unavailability of the network data from the UK (but is freely available from SBB). The intention was to perform all problems on the UK network.

9.1.1 Rail Data Feeds

A framework was setup to listen for all update data transmitted from Network Rail, this data is then manipulated to be indexed into a database from which it can be recalled at a later date. An API was then setup to conveniently retrieve data from the database without explicitly giving database connection details to all end users (which would be a massive security problem). The Darwin implementation is limited to Huxley and performs REST calls to a self-hosted Huxley instance.

9.1.2 GA TSP

A framework for representing a Genetic Algorithm is defined but not fully explored, defining the genetic operators and genotype representation for being able to solve a TSP are defined, as well as hard and soft constraints. A toy example was created in python, it takes an input matrix of travel times and a representation of the network. The framework allows for larger representations that those used in the toy example.

9.1.3 RL TSP

The Reinforcement Learning algorithm defined presents the largest research area of the project, using the instance data from SBB Challenge, the input data was described. A discrete event simulation program was created in Python, and is how the simulation is run, the reinforcement learning algorithm is introduced on making decisions on which train should be stopped in the event of a collision. The Q-Table starts the simulation knowing nothing, and is run for 30 minutes to find the best possible feasible solution by constantly learning how to deal with situations better.

9.1.4 Manchester Analysis

A short analysis was performed on the Manchester Area, Salford Crescent to Manchester Victoria to Manchester Piccadilly. Analysing Utilisation rates of junctions it was clear that there are too many trains in the current timetable and suggestions were made to improve performance. Service suggestions including a new platforming algorithm at Manchester Oxford Road and routing suggestion for Manchester Piccadilly to Manchester Victoria Services. Infrastructure suggestions included an additional track from Manchester Oxford Road to Manchester Piccadilly, allowing the platforming algorithm applied at Oxford Road could also be applied to Manchester Piccadilly, a grade separation at Ordsall Junction to allow for 4 simultaneous movements, currently theoretically 2 (rarely happens due to services not lining up) and an additional track at Irwell Street Junction to allow

for 3 simultaneous movements (2 from Water Street Jn, an additional from Ordsall Jn).

9.2 Conclusion

9.2.1 Research Question 1

What benefits can come from observing all trains?

This question is addressed in Chapter 3, the benefits of observing all trains allows delay to be tracked to give a greater understanding of when and where they occur. More importantly, although not covered in this research paper, the largest benefit is the massive number of data points that are saved and are able to be fed into a Machine Learning algorithm, this allows delays to be better understood, as opposed to just tracked. Machine Learning can be used to 'learn' to highlight delayed trains, but once it has 'learned' what a delayed train looks like it can be used to predict which trains will become delayed. Once this is understood, it is possible to make operational decisions to alleviate these delays on a case by case basis such as slowing a train proactively, so that it never comes to a full stop, this would be especially important for freight trains who cannot accelerate as fast. This, however, relies on massive amounts of data, of which solely tracking trains is not enough and other means of data sources would have to be utilised.

9.2.2 Research Question 2

Can monitoring trains give an advantage for identifying areas prone to delay?

As mentioned in the previous question, where a machine learning algorithm can highlight a train that is expected to become delayed, the same principal can be used for sections of rail or combined so that an algorithm can predict when and where a delay will occur on any train in the UK. For live trains, a system can easily highlight any areas that are currently delayed and feed this into a simple statistical model to break down times of day where specific sections are busy however this information is already known and largely unhelpful, not for understanding the root

cause, but how to mitigate the root cause so that delays can be avoided in the future.

9.2.3 Research Question 3

What are the biggest obstacles for Artificial Intelligence in Railway Scheduling?
This question is addressed in Chapter 4, the obstacles are selecting the most applicable AI technique and designing an appropriate model to represent the network. However for Machine Learning techniques, such as Reinforcement Learning, lack of data is the largest problem that faces AI. Generally speaking, more data means more useful data, which is good for model training. The reinforcement learning technique applied in Chapter 6 gives MDP suggestions for movements, however all the instance solutions are not transferable to other instances, and instance 5 was unsolved. Ideally, with enough data, a single model would be applicable to all train networks as it would be able to make the correct decision $> 99.5\%$ of the time

9.2.4 Research Question 4

Can Reinforcement Learning be applied to the Train Scheduling Problem?
Yes, Reinforcement Learning can be applied to the TSP, however probably not in the way implemented in this research. It is likely that the optimal Machine Learning method is a combination of various techniques which are able to utilise the benefits of each technique. The Reinforcement Learning technique applied in Chapter 6 relies on a table of 'decision' values that are updated as the model progresses due to 'learning' how to deal with situations better. A bonus that Reinforcement Learning has, but also could be considered a downside, is that when the solver starts, it has no knowledge of what it needs to solve and only improves based on the in built positive and negative feedback loops

9.2.5 Research Question 5

Why are trains often delayed on the Manchester Corridor?

This question is discussed in Chapter 8, the cause of delays is the high saturation levels on junctions in Manchester. This combined with feeder corridors causing trains to arrive at Manchester slightly delayed, causing a timetable, that is already not robust, to knock on delays even further. It would be massively beneficial for trains on feeder corridors to arrive as close to scheduled time as possible, as these delays are avoidable. Where as delays from too high utilisation are generally unavoidable.

9.2.6 Research Question 6

What can be done to alleviate pressure on junctions in Manchester?

Suggestions for improvement are made in Chapter 8, the purpose of all improvements are to decrease utilisation of certain junctions, either by grade separation as in Section 8.2.4.4 or by platform allocation algorithms in Section 8.2.1. The need for more capacity on Manchester Corridor is a must for an expanding network, an addition is overall modernisation of the victorian infrastructure currently in place, a welcome change would be to upgrade to more advanced signalling methods used in other european countries.

9.3 Future Work

This section summarises potential further work following on from this research, it is split into the 3 major project areas; Rail Data Feeds, GA TSP and ML TSP.

9.3.1 Rail Data Feeds

Rail Data Feeds provides the greater room for expansion, with various possibilities such as extending web app functionality to allow for a better user experience to compare with other widely used train tracking websites. Originally the plan was to create an interactive map with trains displayed on it, this was reduced during the project because it adds no significant value to the overall research piece. The

statistics side has many paths for expansion, the author would like to explore advanced methods to track delays specific to $(location, operator, service, unit)$

9.3.2 Genetic Algorithms TSP

The genetic algorithm application to calculate train times was only developed for the basic model, the framework discussed in Chapter 5 is not applicable to more advanced models with more services or more complex route network. The modifications required to make the framework applicable to advanced models would require a full rework, therefore it is unlikely for any further work to come from this. Especially since the Reinforcement Learning is far more capable than the GA. However, combining the two, a GA 'warm-start' then feeding into a Reinforcement Learning algorithm could give faster convergence if both models were able to learn what is good and what is bad.

9.3.3 Machine Learning TSP

It was decided that the best area of research in Machine Learning was reinforcement learning, the initial further work would be to manually diagnose instance 5 as to why it was not solve-able and construct a heuristic so the model does not get stuck in the future. In addition to that, computation time is relatively long, especially compared to some of the better SBB submissions, implementing multi-thread processing is hard in python and is not in the scope of this research and is why it was not completed, transitioning to C for better memory management could be a future possibility. Furthermore, having a global q-table to make all decisions would be the true test of whether this technique can cope in real life scheduling scenarios for Network Rail, this was not covered in this research as this research is a litmus test for this discipline. Combining the data from Chapter 3 is the next step realistically for a better model, this was not performed in this research due to the complexity of creating a representative network. For Example, Instance 09 is $1,015,973$ lines long, unless Network Rail could provide a similar type of file it is not possible to create this by hand. For this reason, the UK network could not

be used for the RL TSP and as such the data could not be used to enhance the learning.

Bibliography

Best Solver (2018). iquadrat. URL: https://github.com/iquadrat/sbb-train-scheduler (visited on 01/06/2020).

Castillo, Enrique, Gallego, Inmaculada, Ureña, José María, and Coronado, José María (2008). "Timetabling optimization of a single railway track line with sensitivity analysis". In: *TOP* 17.2, pp. 256–287. DOI: 10.1007/s11750-008-0057-0.

Dündar, Selim and Şahin, İsmail (2013). "Train re-scheduling with genetic algorithms and artificial neural networks for single-track railways". In: *Transportation Research Part C: Emerging Technologies* 27. Selected papers from the Seventh Triennial Symposium on Transportation Analysis (TRISTAN VII), pp. 1–15. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2012.11.001. URL: http://www.sciencedirect.com/science/article/pii/S0968090X12001349.

Fischetti, Matteo and Monaci, Michele (Dec. 2015). "Using a general-purpose MILP solver for the practical solution of real-time train rescheduling". In:

Furini, Fabio and Kidd, Martin Philip (2013). "A fast heuristic approach for train timetabling in a railway node". In: *Electronic Notes in Discrete Mathematics* 41, pp. 205–212. ISSN: 1571-0653. DOI: https://doi.org/10.1016/j.endm.2013.05.094. URL: http://www.sciencedirect.com/science/article/pii/S1571065313000978.

Gestrelius, Sara, Aronsson, Martin, and Peterson, Anders (2017). "A MILP-based heuristic for a commercial train timetabling problem". In: *Transportation Research Procedia* 27. 20th EURO Working Group on Transportation Meeting, EWGT 2017, 4-6 September 2017, Budapest, Hungary, pp. 569–576. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2017.12.118. URL: http://www.sciencedirect.com/science/article/pii/S2352146517310153.

Google OR-Tools (2020). Google. URL: https://developers.google.com/optimization (visited on 08/02/2020).

Gosavi, Abhijit, Bandla, Naveen, and Das, Tapas K. (2002). "A Reinforcement Learning Appoach to Airline Seat Allocation for Multiple Fare Classes with Overbooking". In: *IIE Transactions, Special Issue on Advances on Large-Scale Optimization for Logistics, Production and Manufacturing systems* 34.9, pp. 729–742.

Gurobi (2020). Gurobi. URL: https://www.gurobi.com/ (visited on 08/02/2020).

Higgins, Andrew, Kozan, Erhan, and Ferreira, Luis (Mar. 1997). "Heuristic Techniques for Single Line Train Scheduling". In: *J. Heuristics* 3, pp. 43–62. DOI: 10.1023/A:1009672832658.

Hirashima, Yoichi (2011). "A Reinforcement Learning Method for Train Marshaling Based on Movements of Locomotive". In: *IAENG International Journal of Computer Science* 38.3, pp. 242–248.

– (2012). "On Reinforcement Learning Methods for Generating Train Marshaling Plan Considering Group Layout of Freight Cars". In: *IAENG International Journal of Computer Science* 39.3, pp. 239–245.

Hopgood, Adrian Alan (2012). *Intelligent Systems for Engineers and Scientists*. English. 3rd. CRC Press Inc. ISBN: 9781439821206.

Jean-Francois Cordeau Paulo Toth, Daniele Vigo (1998). "A Survey of Optimization Models for Train Routing and Scheduling". In: *Transportation Science* 32.4,

pp. 380–404. ISSN: 00411655, 15265447. URL: http://www.jstor.org/stable/25768836.

JSON (2020). JSON. URL: https://www.json.org/json-en.html (visited on 02/17/2020).

Kaelbling, Leslie (1994). "Associative Reinforcement Learning: Functions in k-DNF". In: *Machine Learning* 15.3.

Kaelbling, Leslie Pack, Littman, Michael L., and Moore, Andrew W. (May 1996). "Reinforcement Learning: A Survey". In: *J. Artif. Int. Res.* 4.1, pp. 237–285. ISSN: 1076-9757.

Kakade, Sham Machandranath (Mar. 2003). "On the Sample Complexity of Reinforcement Learning". An optional note. PhD thesis. University College London.

Kalyanmoy Deb, William M Spears (1997). "Speciation methods". In: *Handbook of Evolutionary Computation* 97.1. URL: https://pdfs.semanticscholar.org/a525/1192091cc6c138cf8010e43d72b9dfb0d022.pdf.

Khadilkar, H. (2019). "A Scalable Reinforcement Learning Algorithm for Scheduling Railway Lines". In: *IEEE Transactions on Intelligent Transportation Systems* 20.2, pp. 727–736. ISSN: 1558-0016. DOI: 10.1109/TITS.2018.2829165.

Lipinski, Eryk (2015). "Mobile Phone App to Track Movement of a Train". In:

Littman, Michael (1994). "Markov Games as a Framework for Multi-Agent Reinforcement Learning". In: *Proceedings of the Eleventh International Conference on Machine Learning, San Francisco, CA*. Morgan Kaufmann, pp. 157–163.

Lu, Shaofeng, Hillmansen, Stuart, Ho, Tin, and Roberts, Clive (June 2013). "Single-Train Trajectory Optimization". In: *Intelligent Transportation Systems, IEEE Transactions on* 14, pp. 743–750. DOI: 10.1109/TITS.2012.2234118.

Michalewicz, Zbigniew (1996). *Genetic Algorithms + Data Structures =*. Springer Berlin Heidelberg.

MiniZinc (2020). MiniZinc. URL: https://www.minizinc.org/ (visited on 08/02/2020).

Network Rail Data Feeds (2020). Network Rail. URL: https://datafeeds.networkrail.co.uk/ntrod/login (visited on 01/06/2020).

Network Rail Operational Rules (2020). Network Rail. URL: https://www.networkrail.co.uk/industry-and-commercial/information-for-operators (visited on 04/03/2020).

Nygren, Erik, Egli, Adrian, Abels, Dirk, Jöckel, Lothar, and Rothen, Lorena (Oct. 2017). "Reinforcement Learning for Railway Scheduling: Overcoming Data Sparseness through Simulations". In: DOI: 10.13140/RG.2.2.22843.11041.

Open Rail Data Wiki (2020). URL: https://wiki.openraildata.com//index.php?title=Main_Page (visited on 01/06/2020).

Petersen, E. R. (1974). "Over-the-Road Transit Time for a Single Track Railway". In: *Transportation Science* 8.1, pp. 65–74. DOI: 10.1287/trsc.8.1.65.

Reillon, Vincent (Jan. 2018). *Understanding Artificial Intelligence*. European Parliamentary Research Service.

Singleton, James (2015). *Huxley*. URL: https://huxley.unop.uk/ (visited on 01/06/2020).

Sivanandam, S.N. and Deepa, S.N. (2008). *Genetic Algorithm Optimization Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-73190-0. DOI: 10.1007/978-3-540-73190-0_7. URL: https://doi.org/10.1007/978-3-540-73190-0_7.

STOMP (2020). STOMP. URL: https://stomp.github.io/ (visited on 02/17/2020).

Strehl, Alexander, Li, Lihong, Wiewiora, Eric, Langford, John, and Littman, Michael (Jan. 2006). "PAC model-free reinforcement learning". In: vol. 2006. DOI: 10.1145/1143844.1143955.

Sutton, Richard S. and Barto, Andrew G. (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press. URL: http://incompleteideas.net/book/the-book-2nd.html.

Thaduri, Adithya, Galar, Diego, and Kumar, Uday (Dec. 2015). "Railway Assets: A Potential Domain for Big Data Analytics". In: *Procedia Computer Science* 53, pp. 457–467. DOI: 10.1016/j.procs.2015.07.323.

Tormos, María, Lova, A., Barber, Federico, Ingolotti, Laura, Abril, M., and Salido, Miguel (Aug. 2008). "A Genetic Algorithm for Railway Scheduling Problems". In: vol. 128, pp. 255–276. DOI: 10.1007/978-3-540-78985-7_10.

Train Schedule Optimisation Challenge (2018). Crowd AI / SBB CFF FFS. URL: https://www.crowdai.org/challenges/train-schedule-optimisation-challenge (visited on 01/06/2020).

Turner, Christopher, Tiwari, Ashutosh, Starr, Andrew, and Blacktop, Kevin (2016). "A review of key planning and scheduling in the rail industry in Europe and UK". In: *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit* 230.3, pp. 984–998. DOI: 10.1177/0954409714565654.

Watkins, Christopher J. C. H. and Dayan, Peter (1992). "Technical Note Q-Learning." In: *Mach. Learn.* 8, pp. 279–292. URL: http://dblp.uni-trier.de/db/journals/ml/ml8.html#WatkinsD92.

Yin, Jiateng, Chen, Dewang, and Li, Lingxi (Dec. 2014). "Intelligent Train Operation Algorithms for Subway by Expert System and Reinforcement Learning". In: *Intelligent Transportation Systems, IEEE Transactions on* 15, pp. 2561–2571. DOI: 10.1109/TITS.2014.2320757.

Zhang, Wei and Dietterich, Thomas G. (1995). "A Reinforcement Learning Approach to Job-shop Scheduling". In: *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*. Orlando, Florida: Morgan Kaufmann Publisher, pp. 1114–1120.

Appendix A

Markov Decision Processes

A.1 Markov Processes

A stochastic process is a sequence of events in which the outcome at any stage depends on a probability; A Markov process is a stochastic process with additional properties:

1. The number of states in finite
2. The outcome at any stage depends only on the previous stage
3. The probabilities are constant

If a vector is defined x_0 which represents the initial state of a system, then there is a matrix M such that the state of the system after one iteration is given by the vector Mx_0. If there are multiple iterations of state vectors (x_0,Mx_0,$^2Mx_0$,...,M^nx_0) this is called a Markov Chain, and M is the transition matrix. Further to this if M is the transition matrix of a Markov Process such that M^n is strictly positive for any n. Then there exists a probability vector x_s such that M_{x_s} such that $M_{x_s} = x_s$ (A Sink Node because $\liminf kM^nx_0 = x_s$ for any x_0). The probability vector x_s is called the steady state vector of the system.

A.2 Markov Reward Processes

With a Markov Process defined, the Markov Reward Process (MRP) extends this notation to include a reward to each state. Popularised in the book by Ronald Howard. However, in modern day, MRP's are rarely used in favour of Markov Decision Processes where the rewards attained can impact future decisions

A.3 Markov Decision Processes

Markov Decision Processes (MDP) make up the basis of Reinforcement Learning, and as described in the previous section the reward-observation process allows improvements. An MDP M is a $5x1$ array (S,A,T,R,γ), where S is the state space, A is the action space, T is a transition function ($T : S \times A \times S \to \mathbb{R}$), R is a reward function ($S \times A \to \mathbb{R}$) and γ is a discount factor ($0 \leq \gamma < 1$). Letting S denote the number of states and A denote the number of actions. When at state s, performing action a will receive reward r (which has expectation $R(s, a)$) and is taken to state s' with probability $T(s'|s, a)$. The policy is a strategy for choosing actions; a stationary policy produces actions bases solely on the current state. It will be assumed that all rewards are in the interval $0 < r < 1$. Define a policy π, let $V_M^\pi(s)(Q_M^\pi(s, a))$ denote the discounted, action-value(here-in called the Q-Value) function for π in M from state s. If T is a positive integer, let $v_M^\pi(s, T)$ denote the T-step value function of policy π. Specifically, $V_M^\pi(s) = E\sum_{j=1}^{\infty} \gamma^{j-1} r_j$ and $V_M^\pi(, Ts) = E\sum_{j=1}^{T} \gamma^{j-1} r_j$ where $[r_1, r_2, \dots]$ is the reward sequence generated by following policy π from state s. These expectations are taken overall possible infinite paths the agent may follow. The optimal policy is denoted π^* and has value functions $V_M^*(s)$ and $Q_M^*(s, a)$.

A.4 Partially Observable Markov Decision Processes

A generalisation of a normal Markov Decision Process, A Partially Observable Markov Decision Process (POMDP) is where the MDP is unable to see the underlying state. Instead relying of a probability distribution over the set of possible staes based on observations. Although this method is not used in later work, this method is extremely useful to understand the MDP applied to Reinforcement Learning. A POMDP M is a $7x1$ array (S,A,T,R,ω,O,γ), where S is the state space, A is the action space, T is a transition function ($T : S \times A \times S \to \mathbb{R}$), R is a reward function ($S \times A \to \mathbb{R}$), ω is a set of observations, O is a set of conditional observation properties and γ is a discount factor ($0 \leq \gamma < 1$). These definitions are the same as in MDP, with the addition of ω and$O$. When the state is transitioned to s' the agent recieves an observation o which depends on the state of the environment s' and on the action a, with probability $O(o|s', a)$. A Markovian Belief state would allow a POMDP to be formulated as MDP where every belief is a state and is defined as (B, A, ρ, r, γ) where B is the set of belief states, A is the action space, ρ is the belief state transition function, r is a reward function ($B \times A \to \mathbb{R}$) and γ is the discount factor equal to that in a POMDP.

Appendix B

Latex Listings Language Definition

B.1 JavaScript

```
\lstdefinelanguage{Javascript}
{
  morekeywords=[1]{break, continue, delete, else, for, function, if,
    in,
    new, return, this, typeof, var, void, while, with, require, log},
  % Literals, primitive types, and reference types.
  morekeywords=[2]{false, null, true, boolean, number, undefined,
    Array, Boolean, Date, Math, Number, String, Object},
  % Built-ins.
  morekeywords=[3]{eval, parseInt, parseFloat, escape, unescape},

  sensitive,

  commentstyle=\color[rgb]{0.9,0.1,0.1},
  keywordstyle=\color[rgb]{0,0.5,0},
  keywordstyle=[2]\color{blue},
  stringstyle=\color{orange},

  tabsize=2,
  frame=none,
```

```
  numbers=left,
  stepnumber=1,
  numberstyle=\tiny,
  numbersep=5pt,

  xleftmargin=0pt, % numbers will be in the margins!
  columns=fixed, % same width for all characters

  morecomment=[s]{/*}{*/},
  morecomment=[l]//,
  morecomment=[s]{/**}{*/}, % JavaDoc style comments
  morestring=[b]',
  morestring=[b]"

  showstringspaces=false,
  mathescape=true,
  breaklines=true,

  breakatwhitespace=true,
  breakindent=10pt, % was: 20pt
  moredelim=**[is][\color{Melon}]{@}{@},
  escapeinside={{<@}{@>}}
}
```

Listing B.1: 'Javascript Listings Definition'

B.2 MiniZinc

```
\lstdefinelanguage{Mzn}
{
  morekeywords={
  array, par, var, opt, constraint, solve, satisfy, minimize,
  maximize, output, include, let, in, set, of, if, then, else, endif,
  ann, annotation, bool, enum, float, int, string, where, function,
  predicate, true, false, not, assert, trace, any, list, op,
  record, test, tuple, type},

  keywords=[2]{
```

```
11  forall, sliding_sum, symmetry_breaking_constraint,
12  implied_constraint, redundant_constraint, all_different,
13  alldifferent, alldifferent_except_0, alldiff, element, circuit,
14  subcircuit, dpath, card, bool2int, inter, count, regular, table,
     xor,
15  exists, xorall, iffall, clause, intersect, diff, symdiff, subset,
16  superset, concat, join, length, int_lt_reif, int_lin_eq_reif,
17  bool_imply, array_bool_or, all_different_int, int_ne, at_least,
18  at_most, exactly, count_eq, count_leq, count_geq, count_gt, nvalue,
19  bin_packing, bin_packing_capa, bin_packing_load, diffn,
20  global_cardinality, global_cardinality_closed,
21  global_cardinality_low_up, global_cardinality_low_up_closed,
22  inverse, cumulative, disjunctive, decreasing, increasing,
23  strictly_decreasing, strictly_increasing, sort,
24  arg_sort, value_precede, value_precede_chain, lex_less, lex_lesseq,
25  lex_greater, lex_greatereq, lex2, strict_lex2, int_lin_eq,
     all_equal,
26  bool_lin_eq, knapsack, partition_set, member, reverse, among,
     in_set,
27  % from ???:
28  abort, abs, acosh, array_intersect, array_union, array1d, array2d,
29  array3d, array4d, array5d, array6d, asin, atan, bool2int, card,
30  ceil, concat, cos, cosh, dom, dom_array, dom_size, fix, exp, floor,
31  index_set, index_set_1of2, index_set_2of2, index_set_1of3,
32  index_set_2of3, index_set_3of3, index_set_6of6, int2float, is_fixed
     ,
33  join, lb, lb_array, length, ln, log, log2, log10, min, max, pow,
34  product, round, set2array, show, show_int, show_float, sin, sinh,
35  sqrt, sum, tan, tanh, trace, ub, ub_array, xor, in, subset,
36  superset, union, diff, symdiff, intersect, div, mod,
37  % annotations:
38  is_defined_var, output_var, var_is_introduced, defines_var,
39  promise_total, bounds, domain, bool_search, int_search, seq_search,
40  set_search, input_order, first_fail, anti_first_fail, smallest,
41  largest, occurrence, most_constrained, max_regret, indomain_min,
42  indomain_max, indomain_middle, indomain_median, indomain,
43  indomain_random, indomain_split, indomain_reverse_split,
44  indomain_interval, outdomain_max, outdomain_median, outdomain_min,
```

```
45    outdomain_random, complete
46    },
47
48    sensitive,
49    commentstyle=\color[rgb]{0.9,0.1,0.1},
50    keywordstyle=\color[rgb]{0,0.5,0},
51    keywordstyle=[2]\color{blue},
52    stringstyle=\color{orange},
53
54    tabsize=2,
55    frame=none,
56    numbers=left,
57    stepnumber=1,
58    numberstyle=\tiny,
59    numbersep=5pt,
60    xleftmargin=0pt, % numbers will be in the margins!
61    columns=fixed, % same width for all characters
62
63    morecomment=[l]{\%},
64    morestring=[b]",
65    showstringspaces=false,
66    mathescape=true,
67    breaklines=true,
68
69    breakatwhitespace=true,
70    breakindent=10pt, % was: 20pt
71    moredelim=**[is][\color{Melon}]{@}{@},
72    escapeinside={{<@}{@>}}
73  }
```

Listing B.2: 'MiniZinc Listings Definition'

Appendix C

Trains on Manchester Corridor

Operator	Class	Type	Carriages	
			No	Length
NT	142	DMU	2	30.5
	144	DMU	2	30.19
	144	DMU	3	45.28
	150	DMU	2	40.12
	153	DMU	1	23.21
	155	DMU	2	46.42
	156	DMU	2	46.06
	158	DMU	2	49.42
	158	DMU	3	69.63
	170	DMU	2	47.22
	170	DMU	3	70.83
	195	DMU	2	48.05
	195	DMU	3	71.40
	319	EMU	4	79.32
	321	EMU	4	79.80
	322	EMU	4	79.80
	323	EMU	4	93.48
	331	EMU	3	71.40
	331	EMU	4	94.75
	333	EMU	4	94.96

Table C.1: Northern Rail Trains

Operator	Class	Type	Carriages	
			No	Length
TPE	185	DMU	3	71.28
	350	EMU	4	81.60
	68	DMU	5	30.50
	Nova 1	BMU	5	130
	Nova 2	EMU	5	118
	Nova 3	Loco	5	111.85

Table C.2: Trainspennine Express Trains

Operator	Class	Type	Carriages	
			No	Length
EMR	153	DMU	1	23.21
	156	EMU	2	46.06
	158	DMU	2	69.63

Table C.3: East Midlands Regional Trains

Operator	Class	Type	Carriages	
			No	Length
TFW	150	DMU	2	40.12
	153	DMU	1	23.21
	158	DMU	2	69.63
	175	DMU	2	47.42
	175	DMU	3	70.45

Table C.4: Transport For Wales Railway Trains