
Improving railway operations
through the integration of
macroscopic and microscopic modelling
with optimisation

By
Silvia Umiliacchi

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Electronic, Electrical and Systems Engineering

College of Engineering and Physical Sciences

The University of Birmingham, UK

November 2016

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

Abstract

Britain's railway industry is implementing the vision of the system in the next thirty years, as outlined in the Rail Technical Strategy (2012); the main objectives to achieve are: carbon and cost reduction, capacity increase and customer satisfaction.

The timetable design process is identified as a key enabler of the strategy's implementation. The current method in use is considered as a lengthy process with little computer support and optimisation.

This study tries to overcome the outlined weaknesses of the existing method by proposing a more automated process in which the optimisation of a timetable is a properly design stage.

The method has been applied to minimise the total energy consumption of five trains on the Aberdeen-Inverness line, while meeting operational and safety constraints. The results showed a reduction in the total energy consumption of 7%, while the average train total journey time is increased by 1% in comparison with the initial schedule.

Acknowledgements

I would like to thank my supervisors, Prof. Clive Roberts, Prof. Felix Schmid and Dr Meena Dasigi for their support and advice throughout my research.

I also thank Network Rail and the Engineering and Physical Sciences Research Council (EPSRC) for funding my research.

I am very grateful to Dr David Kirkwood for his precious help with setting up and developing the BRaVE simulator and to Dr Gemma Nicholson for her helpful suggestions.

Thanks also to my friends in Wesley House, with whom I have had a very nice time.

Finally, I would like to thank my family and my boyfriend for their love and encouragement that gave me the strength to carry out this work.

Contents

1	Introduction	1
1.1	Background	1
1.2	Aims	3
1.3	Contributions	4
1.4	Thesis structure	5
2	Timetabling process and necessary stages	6
2.1	Introduction to the timetabling problem and its classification	6
2.2	Long-term planning process in Britain and possible improvements . .	10
2.3	Simulation	11
2.3.1	Macroscopic simulation software	13
2.3.2	Mesosopic simulation software	14
2.3.3	Microscopic simulation software	14
2.3.4	Simulator choice	15
2.4	Timetable evaluation	16
2.5	Energy optimisation for timetables	20
2.6	Component Integration for Timetable Design	22
2.7	Summary	25
3	Framework description	27
3.1	Macroscopic and microscopic simulation	28
3.2	Quality of Service - QoS assessment	31

3.3	Optimisation	33
3.4	Closed-loop approach	34
4	Framework Implementation	35
4.1	Macro simulation	35
4.1.1	Viriato	35
4.2	Micro simulation	39
4.2.1	BRaVE	40
4.3	Interface between Viriato and BRaVE	41
4.4	Viriato and BRaVE comparison	43
4.5	Timetable assessment	45
4.6	Interface between evaluation and optimisation stages	47
4.7	Optimisation algorithm	49
4.7.1	Brute force algorithm	49
4.7.2	Genetic algorithm	49
5	Case study: Aberdeen-Inverness Railway Timetable Design	55
5.1	Process deployment	57
5.1.1	Viriato model definition	57
5.1.2	Viriato conflict detection and resolution	59
5.1.3	BRaVE model definition	62
5.1.4	BRaVE conflict detection and resolution	62
5.1.5	BRaVE logging functionality	64
5.1.6	Quality of service assessment	65
5.1.7	Optimisation	68
5.1.8	Loop over the process	73
5.2	Numerical Results	73
5.3	Summary	76

6	Conclusions And Further Work	77
6.1	Achievements	77
6.2	Recommendations	78
6.3	Further work	79
	Appendices	90
A	Optimisation models and algorithms	91
B	Pseudo code of the main parts of the framework	101

List of Figures

2.1	Types of timetabling	9
2.2	Timetabling process	10
3.1	Method components	27
3.2	Models	30
3.3	Quality of service framework (Lu et al., 2013a)	32
3.4	Closed-loop process	34
4.1	Framework software components	35
4.2	Viriato's architecture - (SMA and Partner, 2009)	36
4.3	Example of a macroscopic model in Viriato	38
4.4	Example of timetable graph in Viriato	39
4.5	Example of infrastructure model in BRaVE	42
4.6	Example of timetable in BRaVE	43
4.7	BRaVE - Simulation data	45
4.8	Evaluation - Departure time matrix	46
4.9	Evaluation - Journey time matrix for service S1	46
4.10	Evaluation step - Graphs	47
4.11	Example of XML structure	48
4.12	Genetic algorithm implementation	50
4.13	Genetic algorithm - Initialisation procedure	51
4.14	Example of application of the single-point crossover method	53

4.15	Mutation operator	53
5.1	Case study - Aberdeen-Inverness railway line	55
5.2	Case study - Infrastructure layout of a stretch of the Aberdeen-Inverness railway line (Network Rail, 2014)	56
5.3	Viriato - Definition of the Aberdeen-Dyce section	58
5.4	Viriato - Section model construction	58
5.5	Viriato - Definition of the number of platforms and corresponding length at Inch station (Network Rail, 2014)	59
5.6	Viriato - Definition of entry/exit speed limits at Inch station (Network Rail, 2014)	59
5.7	Viriato - Node model construction	60
5.8	Viriato - Train route and schedule definition	60
5.9	Viriato - Timetable with the additional train	61
5.10	Viriato - Conflict detection	61
5.11	Viriato - Timetable after the first optimisation	62
5.12	Import of a Viriato model into BRaVE	63
5.13	BRaVE model	63
5.14	BRaVE simulation	64
5.15	BRaVE - Example of warning issued	64
5.16	BRaVE - Logging simulation data	65
5.17	BRaVE - Log file	66
5.18	Evaluation - Log file import	67
5.19	Evaluation - Partial journey time matrix	67
5.20	Evaluation graphs	68
5.21	Evaluation - XML file containing the technical data on the railway network under consideration	69
5.22	Train trajectory	70
5.23	Visual representation of the optimisation problem	72

5.24	Optimisation - Fitness function convergence	72
5.25	Final timetable when using cost function (5.1)	75
5.26	Final timetable when using cost function (5.2)	75
6.1	Method components	77
6.2	Method schematic	78

List of Tables

2.1	Features of the method for constructing a timetable	11
2.2	Available macroscopic simulators	13
2.3	Available mesoscopic simulators	14
2.4	Available microscopic simulators	15
2.5	Simulator comparison	15
2.6	Evaluation method classification according to timetable periodicity and model accuracy	17
2.7	Proposed indicators when the perspective of different stakeholders is taken into account	19
2.8	Timetable energy optimisation methods	21
3.1	Macroscopic and microscopic constraints	29
3.2	Key performance indicators	33
4.1	Input to Viriato	38
4.2	Infrastructure data in a BRaVE model	42
4.3	Train data in a BRaVE model	43
5.1	Train technical data	56
5.2	GA parameters	71
5.3	Penalty values depending on the train journey time deviation	74
5.4	Optimisation results with different cost functions	76

A.1	Model variables	92
A.2	Model parameters	92
A.3	Model comparison	97

Chapter 1

Introduction

1.1 Background

Britain's railway industry aims to deliver a better service in order to meet customers' needs, while reducing costs. To this end, a Rail Technical Strategy was devised in 2012, in which a vision of how the British railway system will look in the next thirty years is outlined (TSLG, 2012). The main objectives of the strategy can be summarised in 4Cs: carbon reduction, cost reduction, capacity increase and customer satisfaction. The first two objectives will be achieved mainly by means of electrification, asset specification, energy storage and smart grid technologies, whereas capacity will be increased by developing traffic management systems that have a precise knowledge of train location, speed, braking and load, and by improving asset management, thanks to intelligent maintenance that provides accurate timely information for condition-based intervention. Finally, customer experience will be improved by providing passengers with seamless door-to-door journeys, accurate information and no need for queues or physical barriers at stations.

FuTRO, Future Traffic Regulation Optimisation, is a programme that was launched in 2013, with the aim of identifying technologies, practices, and systems to support the development of intelligent traffic management systems. The essence of FuTRO is an innovative, overarching system for the management of the time, speed, and position dimensions of railway operations to improve their efficiency (RSSB, 2013).

In order to meet its target, the FuTRO programme includes work on driver support system, train location on the railway network and use of shared, open access ontologies and linked data.

In more detail, as found in (Reliable Data Systems, 2014), driver support system is about providing the driver's display with real-time information on train position and route. This will bring benefits to costs, performance during disruptions and safety. This is because the driver will not need extensive training and will make fewer mistakes. Train location, instead, concerns the development of a system that provides accurate information on train position on the network without the need for infrastructure equipment or GPS data (GOBOTiX Limited, 2014). This will help with optimising railway operations. Finally, ontologies and linked data regard making data self-describing to facilitate access to information resources and enable their easy integration and combined usage (Tutcher et al., 2013).

Timetable optimisation is also included; in fact, the British timetable is currently manually constructed and little consideration is given to its ability to recover from disturbances, energy consumption and connectivity between key services (RSSB, 2013).

During 2015, another project was started, the Digital Railway Project. This is a cross-industry programme to support passenger growth and bring about financial saving by accelerating the digital modernisation of the British railway. The scheme aims to enable more trains to run on the network, offering greater reliability and better connections. This will be achieved by upgrading the signalling system, developing a traffic management system that has accurate information on train position and digital decision support systems that will provide the information needed to drivers and passengers. Finally, more sophisticated and automated design and planning tools will enable an efficient development of a more flexible timetable.

1.2 Aims

Both the FuTRO programme and the Digital Railway Project identify the improvement in the timetable design process as a key enabler of the Rail Technical Strategy's implementation. The current method in use is considered as a lengthy process with little computer support and optimisation. In more detail, the current process to construct the British railway timetable for the short- or long-term employs the following software packages: Voyager Plan, ITPS (Integrated Timetable Planning System) and RailSys. These systems are not integrated and little consideration is given to timetable optimisation in terms of robustness, energy consumption and connectivity (RSSB, 2013).

This thesis has the following aims:

- developing a method to improve timetables that makes extensive use of integrated computer systems, requiring low intervention by timetable planners.
- providing an optimisation architecture within the framework that can be used to address the optimisation needs of a given railway system.

1.3 Contributions

The author believes that this thesis brings the following contributions to the state-of-the-art timetable construction's process:

- development of a general and scalable framework that can be used in different railway systems to produce feasible and optimised timetables according to the specific needs,
- integration of the software tools that implement the framework,
- low level of manual intervention

1.4 Thesis Structure

This thesis is organised as follows:

- Chapter 1 describes the context of this research, its aims and contributions.
- Chapter 2 gives an overview of the timetabling process and the proposed method. First, it illustrates the timetabling problem and its possible classifications; then, it describes the long-term planning process in Britain and the scope for improvements. Finally, the method components and the relative literature review are discussed.
- Chapter 3 describes the proposed framework. This is built upon the following desired features: automation, integration, feasibility, optimisation, scalability and generality.
- Chapter 4 focuses on the implementation and integration of the different method components with the outlined desired method capabilities in mind.
- Chapter 5 describes the framework's implementation for a selected case study and discusses the results obtained.
- Chapter 6 provides a critical analysis of the work that has been done and outlines possible extensions and improvements.

Chapter 2

Timetabling process and necessary stages

The current process used in the United Kingdom for developing timetables, as described by Chen and Roberts (2012), aims to produce a non-conflicting schedule over a route, so that trains obeying the defined schedule can run undisturbed. The process complexity depends on factors such as network size, traffic heterogeneity and operational rules. The timetabling process is formed of three stages: long-term planning, short-term planning and ad-hoc planning. In the following sections, first, the timetabling problem and its possible categorisation are described; then, the long-term planning stage of the timetabling process in Britain is explained; finally, the components that characterise the proposed method are illustrated together with the related literature.

2.1 Introduction To The Timetabling Problem And Its Classification

Given a set of stations, a set of resources and a train service intention, the train timetabling problem (*TTP*) concerns the assignment of a departure and an arrival time to each train at each station while meeting operational and safety constraints (Panou et al., 2013).

There are different types of timetabling categorisation; a first classification distinguishes between static and dynamic timetabling. The former aims to produce a

timetable that does not change over time, whereas the latter is about making changes to an existing timetable to recover as quickly as possible from possible disturbances during real-time operations.

Another classification distinguishes between a nominal and a robust version of the timetabling process. Cacchiani and Toth (2012) suggest the following definition of the two versions. A nominal train timetabling problem consists of determining a timetable for a set of trains over a given network or a single one-way line satisfying track capacity constraints and optimising an objective function that meets the railway company's needs. This can be, for example, the maximisation of passenger satisfaction or the implementation of TOCs' proposed schedules (Cacchiani and Toth, 2012). The robust version of the problem aims instead to find a schedule that, in case of network disruption, reduces the size of delay propagation. Caprara et al. (2014) provide the following definition for nominal and robust optimisation event-based problems: the nominal optimisation problem consists of deciding upon the property values of each event according to a given cost function and constraints. The robust version of the problem consists of assigning property values to each event so that the corresponding network minimises the total delay propagation. The nominal and robust terms are used in an event-based context, but they go in the same direction as (Cacchiani and Toth, 2012).

A third classification refers to the concepts of strategic, tactical and operational planning. Marinov et al. (2013) state that, in rail operations, three levels of management can be distinguished:

- **Strategic level:** long-term planning. The strategic goals are defined, which include major changes to the infrastructure.
- **Tactical level:** medium-term planning. All the plans and timetables are developed. Capacity and performance analysis are also carried out.

- **Operational level:** short-term planning. The plans and timetables are implemented on a day-to-day basis to provide the service.

In the above classification, strategic planning is depicted as a long-term phase, in which decisions about policies are taken. Tactical planning, instead, concerns the development of the plans. Finally, the operational stage regards short-term changes to the timetable on a daily basis. Watson (2001) provides similar definitions to Marinov's, but with the difference that in the tactical stage not all the resources are considered fixed. These are summarised below (Watson, 2001):

- **Strategic planning:** stage at which changes to the infrastructure can be considered to increase for example the network capacity.
- **Tactical planning:** in this phase, the infrastructure tends to be fixed, but the mobile resources (rolling stock and people) can be varied in quantity, quality and intensity of operation.
- **Operational planning:** stage at which real-time perturbations are taken into account.

Maroti (2006) relates strategic, tactical and operational planning to the timetabling process, as follows:

- **Strategic planning:** stage concerning decision making several years in advance, in which the desired quality of service is specified and the resource availability is checked. The basic shape of the timetable is drawn.
- **Tactical planning:** phase with a two-months to one-year horizon, which takes the defined line plan and service demand as input and outputs a timetable for a generic week of the year. Tactical planning also has, in part, an operational character, since its products take many real details into account.
- **Operational planning:** phase with a 3-day to two-months horizon. The generic week plan produced in the tactical stage is adjusted in response to the

specific demands of the particular week, such as the need of extra trains for occasional events or infrastructure unavailability due to maintenance works.

Maroti (2006) adds a further stage to the classification: short-term planning, which has an horizon of at most three days and includes real-time reactions to the latest developments.

The last classification is related to the time horizon of planning. Within the tactical planning, long-term planning is defined by Watson (2001) as a process that produces timetables and resource plans that are in operation for a number of months, commonly between three and twelve, whilst short-term planning is a process where changes are made to the long-term plan to cope with supply or demand fluctuations. Chen and Roberts (2012) add a further stage, ad-hoc planning, which concerns the regulation of a timetable due to perturbations that prevent use of the daily timetable.

The following table summarises the different classifications of the timetabling problem:

Phase	<i>Preliminary timetabling</i>	<i>Static timetabling</i>	<i>Dynamic timetabling</i>
Timing	<i>Long term planning</i>	<i>Short term planning</i>	<i>Ad-hoc planning</i>
Method	<i>Strategic planning</i>	<i>Tactical planning</i>	<i>Operational planning</i>
Properties	<i>Feasible</i>	<i>Nominal</i>	<i>Resilient (stable, robust and recoverable)</i>

Figure 2.1: Types of timetabling

Preliminary timetabling has been inserted in the table to identify the stage in which a draft timetable is produced or its requirements are defined. A resilient timetable can resist small delays thanks to its design or by means of active rescheduling actions or operational management measures.

2.2 Long-Term Planning Process In Britain And Possible Improvements

This thesis focuses on long-term planning, which in Britain produces two timetables per year. The current British system for long-term planning requires Train Operating Companies (TOCs) to input bids for track access into the Voyager plan system. This data is electronically exchanged with the Integrated Timetable Planning System (ITPS) used by the Infrastructure Manager (IM) Network Rail. ITPS uses the sectional running times agreed with the TOCs to check for timetable conflicts concerning track occupation. The system relies on the experience of timetable planners. The operation of the produced timetable is then simulated using RailSys to analyse for example its behaviour in presence of delays. The process is summarised in figure 2.2.

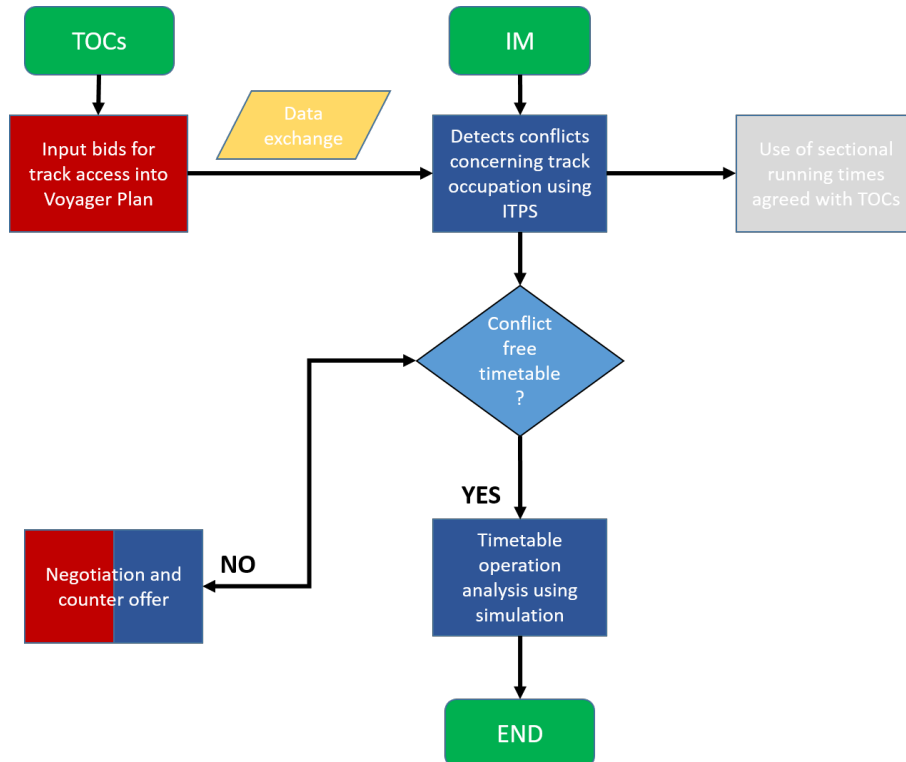


Figure 2.2: Timetabling process

The current method shows scope for improvement in terms of automation and integration: ITPS is only partially automated and not integrated with the RailSys simulator. Moreover, the level of timetable optimisation with regard to robustness, energy consumption and connectivity is limited (RSSB, 2013). Finally, ITPS cannot ensure that trains run undisturbed, since it considers only sectional running times. This study brings some of these improvements to the current timetabling process. The properties that characterise the novel method are defined in table 2.1.

Feature	Definition
Automation	Absence of human intervention in the process
Generality	Ability to deal with different scenarios, for example diverse railway networks
Feasibility	Generation of a timetable that allows trains to run undisturbed
Integration	Use of combined modules
Optimisation	Improvement of the offered railway service with regard to the specific needs of the network under consideration
Scalability	Capability of dealing with large-scale problems

Table 2.1: Features of the method for constructing a timetable

In this thesis, the novel method is applied to reduce the operations' energy consumption. The following sections illustrate what components should be part of the method and why they should be integrated.

2.3 Simulation

Simulation is vital for validating timetables before these are used on the railway network. Feasibility assures that trains will run without interruptions or unnecessary braking and re-acceleration (Goverde et al., 2015).

A simulation model of a railway network consists of infrastructure, vehicles and a timetable. The following sections describe available simulation software according to the level of detail at which the railway infrastructure is represented. The infrastructure is drawn as a graph, in which nodes represent locations on the railway network, i.e. signals, points or timing points, and links are connections between nodes.

There exist three types of infrastructure model:

- macroscopic
- microscopic
- mesoscopic

A macroscopic model gives an abstract view of the infrastructure; nodes represent whole stations or junctions and are connected by links. This model is usually preferred for strategic planning or routing problems, because it does not have accurate information on the track layout or the signalling system and therefore cannot provide accurate results.

A microscopic model, instead, represents the system accurately at the level of block sections, that is, track sections protected by signals. This type of model can be used for running time calculations, conflict detection and resolution, timetable construction and simulation. It is usually employed for operational planning of railway networks. Finally, a mesoscopic model shows a level of detail that is between a macroscopic and a microscopic model.

In the following sections, software packages that implement the different types of model are discussed.

2.3.1 Macroscopic Simulation Software

An overview of the software packages that implement a macroscopic model of a given railway network is given in table 2.2 and the purpose of each software package is described.

Software package	Scope
NEMO (Kettner et al., 2003)	Evaluation of strategic infrastructure and operational concepts, such as estimation of future traffic volume or the impact of operational changes on the traffic volume on a given network.
PETER (Goverde and Odijk, 2002)	Stability analysis of periodic timetables on given networks. The indicators that are used in the assessment include: cycle times, throughput, stability margins, cumulative recovery times and delay propagation.
FASTA (Putallaz and Rivier, 2004)	Timetable stability analysis of periodic timetables, in particular, delay generation and propagation analysis.
CAPRES (Lucchini et al., 2001)	Development and saturation of periodic timetables at network level to assess the overall capacity.
VIRIATO (SMA and Partner, 2009)	Development and feasibility analysis of timetables.

Table 2.2: Available macroscopic simulators

2.3.2 Mesoscopic Simulation Software

The available software that implements a mesoscopic model of the railway infrastructure is discussed in table 2.3.

Software package	Scope
OnTime (VIA Consulting & Development GmbH, 2016)	Comparing timetables on a given network to predict the variability in punctuality or to assess their robustness in the face of temporary speed restrictions.
SIMUL 8 (Marinov and Viegas, 2011)	Analysis and evaluation of freight train operations.
TTPSW (De Fabris et al., 2014)	Automatic timetable generation.

Table 2.3: Available mesoscopic simulators

2.3.3 Microscopic Simulation Software

The available simulation software that uses a microscopic model of the railway infrastructure is introduced in table 2.4 on the following page.

SIMONE’s main purpose is to analyse a timetable’s robustness. FALKO and RailPlan lack portability: they can be used only on a Windows platform. Finally, the publicly available information on the software package Railsim is limited.

Therefore, only the following software packages are taken into account: BRaVE, OpenTrack, and RailSys. These are compared in terms of scope, integration with other software, flexibility and portability, as summarised in table 2.5 on the next page.

Software	Scope
SIMONE (Middlekoop and Bouwman, 2001)	Analysis of timetable robustness and stability; identification of capacity bottlenecks.
FALCO (SIEMENS, 2007)	Timetable design and validation taking into account possible faults to infrastructure equipment, rolling stock and/or delays.
Railsim (Douglas, 2014)	High-density passenger rail corridors modelling.
BRaVE	Timetable validation and evaluation of railway operations according to the specific needs.
OpenTrack (Nash and Huerlimann, 2004)	Evaluation of infrastructure improvement plans, timetable stability and feasibility.
RailPlan (Trapeze Group, 2016)	Timetable construction, stability analysis and resource scheduling
RailSys (Pouryousef et al., 2015)	Planning of infrastructure and timetable; simulation.

Table 2.4: Available microscopic simulators

Software	Integration	Flexibility	Portability	Reference
BRaVE	RailML and other formats	Available for research purposes	Different computer platforms	Not available
OpenTrack v. 1.3	RailML format	API allows to test new control concepts (e.g. reducing conflicts)	Different computer platforms	(Nash and Huerlimann, 2004)
RailSys v. 10	RailML and other formats	Commercial package	Different computer platforms	(RMCON, 2016)

Table 2.5: Simulator comparison

2.3.4 Simulator Choice

Two simulators are selected for timetable validation: one at a macroscopic level and one at a microscopic level. A combination of the two simulators reduces the

computational time of the planning process. The benefits of combining macroscopic and microscopic models are further discussed in the remainder of this thesis.

At a macroscopic level, Viriato was chosen because it supports the development and feasibility analysis of timetables, whereas the other macroscopic simulation software packages mainly focus on the robustness or the capacity of a timetable.

At a microscopic level, instead, BRaVE was chosen because of its availability for research purposes.

2.4 Timetable Evaluation

Timetables should be assessed in order to establish the quality of service they enable. How performance is evaluated varies, depending on the aims of the railway and country (Siefer, 2008). The main questions that arise when approaching timetables' assessment are:

- What should be assessed?
- Whose interests should be represented?

The following literature is structured by timetable periodicity and accuracy of the infrastructure model (table 2.6 on the following page).

On the one hand, timetable periodicity influences the definition of evaluation indicators and methods (Chen et al., 2016). On the other hand, the accuracy of the evaluation process depends on the level of detail at which the railway system is analysed.

Paper	Timetable periodicity	Model accuracy
Schittenhelm and Landex (2010)		
Sels et al. (2015)	cyclic	Macroscopic
Takagi (2012)		
Jiang et al. (2016)	cyclic	Microscopic
Kunimatsu et al. (2012)		
Chen et al. (2016)	Non cyclic	Macroscopic
Nicholson et al. (2015)	Both	Microscopic
Goverde and Hansen (2013)	Both	Macroscopic and microscopic

Table 2.6: Evaluation method classification according to timetable periodicity and model accuracy

The table is structured in three parts: in the upper part, a cyclic timetable is evaluated at the macro- and microscopic levels, respectively. In the central part, a non-cyclic timetable is studied at a macroscopic level, whereas, in the lower part, both types of timetable are analysed mainly at the microscopic level.

The timetable period depends on the railway network under consideration; possible timetable periods are thirty minutes or one hour.

Macroscopic evaluation models are usually based on a given timetable and input data, while microscopic ones employ simulation to calculate the defined indicators.

Different interests can be represented. Takagi (2012) and Kunimatsu et al. (2012) formulate indicators that represent exclusively the passengers' perspective; they state that passengers are the ultimate customers of the railway, thus, their needs should be met. All the other papers try to take into account the point of view of different stakeholders, which include passengers, infrastructure managers and train operators. The precise indicators that are used for the evaluation are particular to each work. These are summarised in table 2.7 on page 19.

An accurate and general framework is reliable and flexible. The evaluation model proposed by Nicholson et al. (2015) meets these requirements and therefore is selected. Two of the proposed indicators are considered: journey time and energy consumption. Journey time is chosen as one of the indicators that expresses passengers' and infrastructure manager's satisfaction, whereas energy consumption represents the environmental and train operating companies' concerns. The evaluation process developed in this thesis is illustrated in section 3.2 on page 31. It should be noticed that the definition of the two indicators differs from the one suggested by Nicholson et al. (2015).

Paper	Perspective	Indicators
Schittenhelm and Landex (2010)	Passengers, IM, TOCs	Fixed interval service frequency
		Percentage of direct connections
		Transfer waiting time
		Use of dedicated rolling stock, train personnel and tracks
		Travel time
Sels et al. (2015)	Passengers, IM, TOCs	Total expected passenger time
Chen et al. (2016)	Passengers, Transport and	Depend on the group
	vehicle departments, Stations	
Nicholson et al. (2015)	Passengers, IM, TOCs, timetable planners	Transport volume
		Journey time
		Connectivity
		Punctuality
		Resilience
		Energy consumption
Goverde and Hansen (2013)	Passengers, IM, TOCs	Resource usage
		Feasibility
		Stability
		Robustness
		Resilience
		Infrastructure Occupation

Table 2.7: Proposed indicators when the perspective of different stakeholders is taken into account

2.5 Energy Optimisation For Timetables

Timetable optimisation enables more energy-efficient operations.

The design of train arrival and departure times can enable an energy-efficient driving strategy or the synchronisation between accelerating and decelerating trains so that they can exchange energy.

Table 2.8 on the next page summarises recent research on developing an energy-optimised timetable, in which regenerative energy is used (upper part) or not (bottom part). Timetable acts on the train energy consumption by means of sectional running times, dwell times at stations, time supplements and train order. Regenerative braking is mainly applied to metro lines, although in some cases, for example (Li and Lo, 2014b), the method can be extended to high-speed or passenger lines. In terms of infrastructure and rolling stock characteristics, speed limits are usually included in the problem formulation, whereas gradients are often neglected or simplified when regenerative braking is taken into account. Rolling stock features are often simplified, that is, maximum traction and braking forces are considered to be constants or even omitted (Li et al. (2013), Ghoseiri et al. (2004)). This study reveals that a timetable is usually optimised in conjunction with the train speed profile and that including regenerative braking affects the problem's formulation, but not the method itself.

Conforming to the literature, one of the timetable elements that affect energy consumption is considered in this thesis, that is, running times. The addressed optimisation problem is described in chapter 5 on page 55.

Paper(s)	Scope	Network type	Infrastructure layout	Rolling stock	Timetable influence element(s)	Combined with speed profile?
Li and Lo (2014a)	Minimise net energy consumption	Metro	Speed limits, no gradient	Constant traction, braking force	Number of trains, sectional running time, dwell time	Yes
Li and Lo (2014b)	Minimise net energy consumption	Metro	Speed limits and constant segment gradient	Constant traction, braking force	Inter-station running time	Yes
Yang et al. (2015)	Minimise energy consumption and passenger travel time	Metro	Speed limits and gradient	Constant traction, braking force	Dwell time	Yes
Watanabe and Koseki (2015)	Minimise energy consumption with fixed total journey time (one train)	Mainline	No speed limits, no gradient	Constant traction, braking force	Sectional running time	Yes
Su et al. (2013)	Minimise total energy consumption with fixed total trip time	Metro	Speed limits, no gradient	Constant traction, braking force	Inter-station running time	Yes
Xu et al. (2016)	Minimise passenger time and energy consumption	Subway	Speed limits, no gradient	Constant traction, braking force	Running and dwell time	Yes
Chevrier et al. (2013)	Minimise total journey time and energy consumption (one train)	Mainline	Speed limits and gradient	Variable traction, braking force	Sectional running time	Yes
Scheepmaker and Goverde (2015)	Minimise total traction energy consumption	Mainline	Speed limits and gradient	Variable traction, braking force	Running time, supplement distribution	Yes
Li et al. (2013)	Minimise total energy, carbon emission costs and total passenger travel time	Mainline	Speed limits and gradient	Constant speed	Running and dwell time, train order	No
Ghoseiri et al. (2004)	Minimise fuel consumption and total passenger time	Mainline	Speed limits and gradient	Constant speed	Running and dwell time, train order	No

Table 2.8: Timetable energy optimisation methods

2.6 Component Integration For Timetable Design

Integrating simulation, evaluation and optimisation components enables the design of feasible and optimised timetables in an automated way.

Approaches that follow this principle are discussed in:

- Caimi et al. (2011)
- Schlechte (2011)
- Goverde et al. (2015)
- Radtke and Hauptmann (2004)
- Kroon et al. (2009)

Each study is described below.

Caimi et al. (2011) developed a multi-level approach for constructing a feasible periodic timetable on a large railway network. After formally defining the service intention, a macroscopic timetable for one period of time is produced using a simplified model of the given railway system. The resulting arrival and departure times are given as time windows. A timetable is then computed at a microscopic level. Given the complexity of the railway network, the approach distinguishes between condensation and compensation zones. The first type refers to areas around stations, while the second one is about links between the condensation zones. A set of alternatives for each train is computed and stored in a data structure; in condensation zones, these alternatives are routes and starting times for each train, whereas, in compensation zones, they represent different speed profiles for given routes. Constraints are derived and formalised in a mathematical model that is then solved. If the resulting timetable is not feasible, time alternatives are input into the macroscopic level and the process is repeated. Otherwise, the final timetable is extended to a whole day.

The method generates feasible timetables; however, it does not evaluate the offered quality of service during the timetable development.

Schlechte (2011) proposes a bottom-up approach. The author combines several software tools to produce a feasible timetable, in which maximum capacity utilisation is achieved. The framework works as follows: a microscopic model of the railway system is automatically simplified to a macroscopic model in terms of infrastructure and times. The latter is used to generate a timetable that makes optimal use of the available capacity by solving a train path allocation problem. The model is then transformed back into a microscopic one in order to check its feasibility. The authors state that developing a timetable at a macroscopic level allows dealing with large networks more efficiently. The main disadvantage of this method is that if a timetable is not feasible, it is possible that, in order to solve conflicts, all the benefits from the optimisation are lost.

Goverde et al. (2015) overcome the weakness in the work by Schlechte (2011) by iterating the interaction between the micro- and the macroscopic level. Specifically, they propose a performance-based railway timetabling framework to obtain a stable, robust, conflict-free and energy-efficient timetable with acceptable infrastructure occupation and short travel times. The framework works as follows: a microscopic model computes detailed train running and blocking times and aggregates the results into a macroscopic model that contains only the main stations where trains need synchronising and ordering. The macroscopic model computes a network timetable that optimises efficiency and robustness taking into account the constraints set by the microscopic model. The macroscopic timetable is then transformed back to the microscopic model, which is used for conflict detection, infrastructure occupation and stability analysis, given the macroscopic timetable. These micro-macro interactions are repeated until a conflict-free, stable and robust timetable is obtained. Finally, the speed profiles of all the trains on each corridor between main stations are optimised while maintaining the scheduled event times

at the corridor ends. The proposed framework is highly automated; however, it shows two weaknesses: the conflict detection feature does not explicitly address the case in which two trains try to occupy the same block section on single-track lines. Moreover, the feasibility check is performed entirely at a microscopic level; this can be time-consuming for complex railway networks.

Radtke and Hauptmann (2004) limit the interaction between macro- and microscopic models to the initial stage, in which a macroscopic timetable can be automatically converted into a microscopic one by a module integrated into RailSys: the slot search engine. This piece of software can be used to accommodate additional trains into an existing timetable in order of priority; for each train, it searches for a conflict-free and optimised train path. The optimisation concerns a combination of the following parameters: running time, dwell time and time deviation from the scheduled departure time at the origin station. The timetable is then simulated using RailSys in order to assess the related quality of service. In this framework, timetable construction and simulation are integrated. Missing features are the optimisation of the overall timetable, not just the extra trains, and the use of evaluation to guide the optimisation process.

Finally, Kroon et al. (2009) describe the decision support system DONS (Designer of Network Schedules) that has been developed in the Netherlands to construct a periodic timetable with better connectivity. This is made of two modules: CADANS and STATIONS.

CADANS employs constraint programming techniques to generate a feasible timetable. In case this cannot be produced, the unsatisfied constraints are provided and indications are given on how to solve the infeasibility. The timetable planner has to input the modified constraints into CADANS, which tries to find another solution. Once a feasible timetable is obtained, the train arrival and departure times are manually modified in order to optimise the transfer times at specific stations. STATIONS, instead, is a piece of software that calculates the train routes

through stations. DONS can be interfaced to SIMONE (Simulation Model for Networks) to evaluate the robustness of a periodic timetable. The system produces a feasible and optimised timetable; however, the optimisation procedure is limited to connectivity, other important aspects of a railway service such as punctuality or energy consumption are not included.

In the literature, timetables are usually assessed after being optimised. The framework described in this thesis proposes a pre-optimisation assessment of a timetable to establish how this should be optimised. This enables a more flexible and general process that can be applied to railway systems with different features and needs.

2.7 Summary

In this chapter, after introducing the reader to the timetabling problem, the long-term process for constructing a timetable is described. This process is carried out mainly manually, taking several months of work depending on the complexity of the problem, and the resulting timetable may not be feasible. The latter is due to the use of static sectional running times for conflict detection. Finally, the quality of service of a timetable is assessed only at the end of the process and focuses on delays, since these can lead to costs incurred by the infrastructure manager.

In order to overcome the outlined design flaws, a novel method is proposed by the author, in which simulation, evaluation and optimisation functions are combined.

Simulation emulates the behaviour of trains on the railway network and, depending on the accuracy of the railway infrastructure model, it guarantees undisturbed train movements. Three levels of accuracy are identified: macroscopic, mesoscopic and microscopic.

Evaluating a timetable using a configurable set of performance indicators instead allows to understand how the timetable could be modified to improve the quality of

service on the given railway network and whether the application of the proposed changes to the timetable has a positive impact on the quality of service.

Finally, the optimisation procedure is responsible for deciding what modifications should be made to a timetable to achieve a desired quality of service.

The novel framework will shorten the time necessary for producing a timetable from months to weeks and it will output a timetable that enables conflict-free train movements and higher operations' performances.

In the next chapter the framework is described, chapter 4 on page 35 and chapter 5 on page 55 instead focus on the implementation and the results obtained using a case study.

Chapter 3

Framework description

The framework is illustrated in figure 3.1. The architecture is formed of the following components:

- Feasibility analysis at a macroscopic level;
- Feasibility analysis at a microscopic level;
- Quality assessment;
- Optimisation.

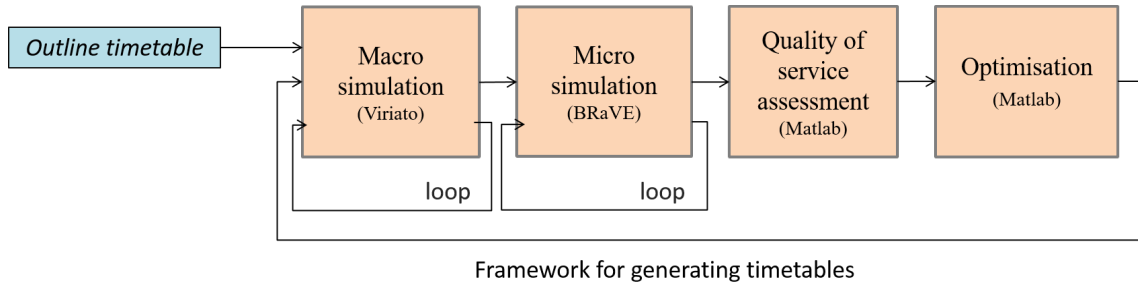


Figure 3.1: Method components

An outline timetable for the railway network is used as the input to the process. This may be an existing timetable, to which modifications are to be made, or a newly created timetable. It is possible at this stage for the proposed timetable not to meet the headway time and infrastructure constraints imposed. The aim of the first step is therefore to enforce the constraints using a macroscopic model of the network.

This is achieved by modifying the timetable until every constraint is satisfied, as highlighted by a loop arrow in figure 3.1 on the preceding page.

The timetable automatically progresses to the second stage of the process, where constraint satisfaction is analysed by simulating the physical behaviour of the trains and modifications are made to the timetable so that it fulfils the given constraints. Another loop arrow in figure 3.1 on the previous page means that the timetable does not enter the third stage until it is feasible.

In the third step, the quality of service provided by the resulting timetable is assessed using a combination of performance indicators.

After its evaluation, the timetable is modified, in the fourth step, by an optimisation algorithm in order to improve the provided quality of service. The optimised timetable may violate headway time and/or infrastructure constraints; therefore, it is input into the first step and the whole process is repeated.

The final timetable is returned when either a maximum number of iterations has been reached or an acceptable quality of service is offered.

In order to automate the process as much as possible, each stage of the method is linked to the following one by an interface. This allows timetable planners to focus on the key aspects of the design process.

The following sections describe each stage of the process in more detail, whereas the implementation is explained in chapter 4 on page 35.

3.1 Macroscopic And Microscopic Simulation

A macroscopic (microscopic) simulator is provided with a draft timetable as input and returns a feasible timetable as output. The simulator analyses the timetable and marks conflicts that occur between trains due to unsatisfied operational or safety constraints. Possible conflicts can be solved by modifying the arrival and/or the departure times of trains at involved stations. The platform used by a train at a

station can also be changed. The conflict resolution process is automated. The constraints that are checked are summarised in table 3.1:

Macroscopic constraints	Microscopic constraints
Headway time between consecutive trains at stations	Headway time between consecutive trains at a block-section level
Separation times at stations (or passing loops)	Arrival time at crossing locations on single-track lines
Access limitations due to the station (or passing loop) layout	-
Capacity at stations (or passing loops)	-

Table 3.1: Macroscopic and microscopic constraints

Headway times between following trains are imposed to ensure that trains can run undisturbed. This constraint is analysed at a macroscopic and a microscopic level. The second constraint, instead, refers to single-track lines, where trains are allowed to meet only at stations or passing loops. Therefore, trains have to arrive at a crossing location at a given time (or time window). This constraint is checked at a microscopic level in order to ensure operationally feasible timetables.

Three more constraints are imposed only at a macroscopic level. The first is regarding the time interval between the arrival of one train and the departure (or arrival) of another train from (at) the same station (or passing loop), when they use the same track, in order to allow for points to be set correctly and the route to be cleared.

The second constraint, instead, refers to the unavailability of tracks inside a station (or a passing loop) in a specific direction of travel due to the station (or the passing loop) layout.

Finally, the last constraint assures that the number of trains in a station (or a passing loop) at a given time does not exceed the maximum available station (or passing loop) capacity.

A timetable is validated at a macroscopic and a microscopic level in sequence, where macro- and microscopic refer to the level of detail in which the railway infrastructure is modelled, as exemplified in figure 3.2.

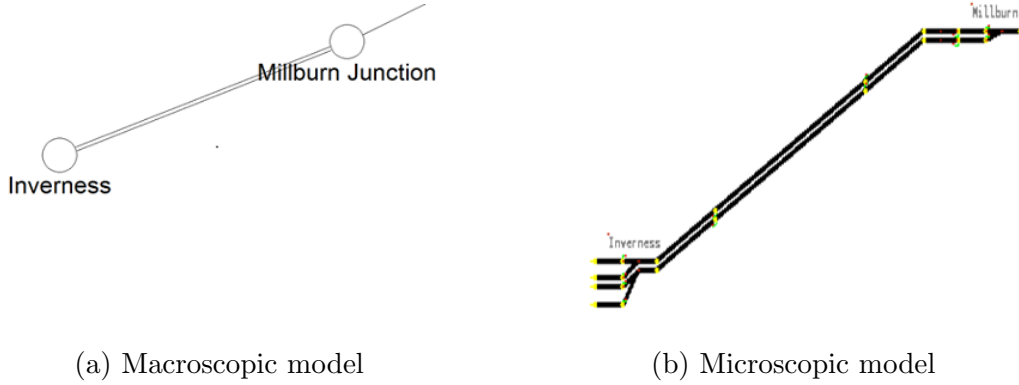


Figure 3.2: Models

A combination of the two models guarantees the feasibility of the resulting timetable and efficiency in performing the analysis on a complex railway network.

A macroscopic simulator cannot ensure safe operations, since it neglects significant aspects of the railway system, but it can flag conflicts at macroscopic points and these conflicts can be analysed independently.

A microscopic simulator, instead, analyses a timetable at a higher level of detail of the railway infrastructure; thus, it is time-consuming. Moreover, in order to analyse a specific conflict, the simulator has to run the timetable until the time when the conflict occurs in order to reproduce the problem.

Using only a microscopic simulator for timetable feasibility is not very efficient when dealing with a complex railway network because possible conflicts are flagged only at the time they occur; consequently, the resolution of one conflict may create new conflicts in other parts of the railway network that were not foreseen, leading to a longer computational time to obtain a feasible timetable. Even though a macro-

scopic simulator cannot produce a feasible timetable due to the granularity of its railway infrastructure model, it can help the microscopic simulator to overcome its limitations by identifying all the macroscopic conflicts on the given network at once and by analysing the impact of a resolution strategy for one conflict on the entire railway network. A mesoscopic simulator was not chosen because it cannot guarantee conflict-free train movements since train detection systems are not included in the model and the location of signals on the track is not accurate.

3.2 Quality Of Service - QoS Assessment

The feasible timetable from the microscopic simulator is input into the evaluation stage, where it is assessed in terms of the quality of service that it enables on the railway network being considered. The framework proposed by Lu et al. (2013a) is used for the assessment; this identifies the factors that influence the performance of a given railway system (inputs) and also the indicators that can be used to measure it (outputs). Figure 3.3 on the next page shows the structure of the framework, in which the design of a timetable is one influencing factor.

Two performance indicators are considered in this thesis to measure the quality of service of a timetable: journey time and energy consumption. Their definitions are provided in table 3.2. Other possible indicators can be: accommodation, resource usage, passenger comfort, etc., see figure 3.3.

As shown in figure 3.3, the timetable component together with the selected indicators are highlighted by a blue rectangle.

At this stage, the indicators that should be part of the optimisation cost function are selected and time constraints that trains have to respect are formulated. Both the cost function and the constraints are input into the optimisation stage with technical data on the railway network and trains.

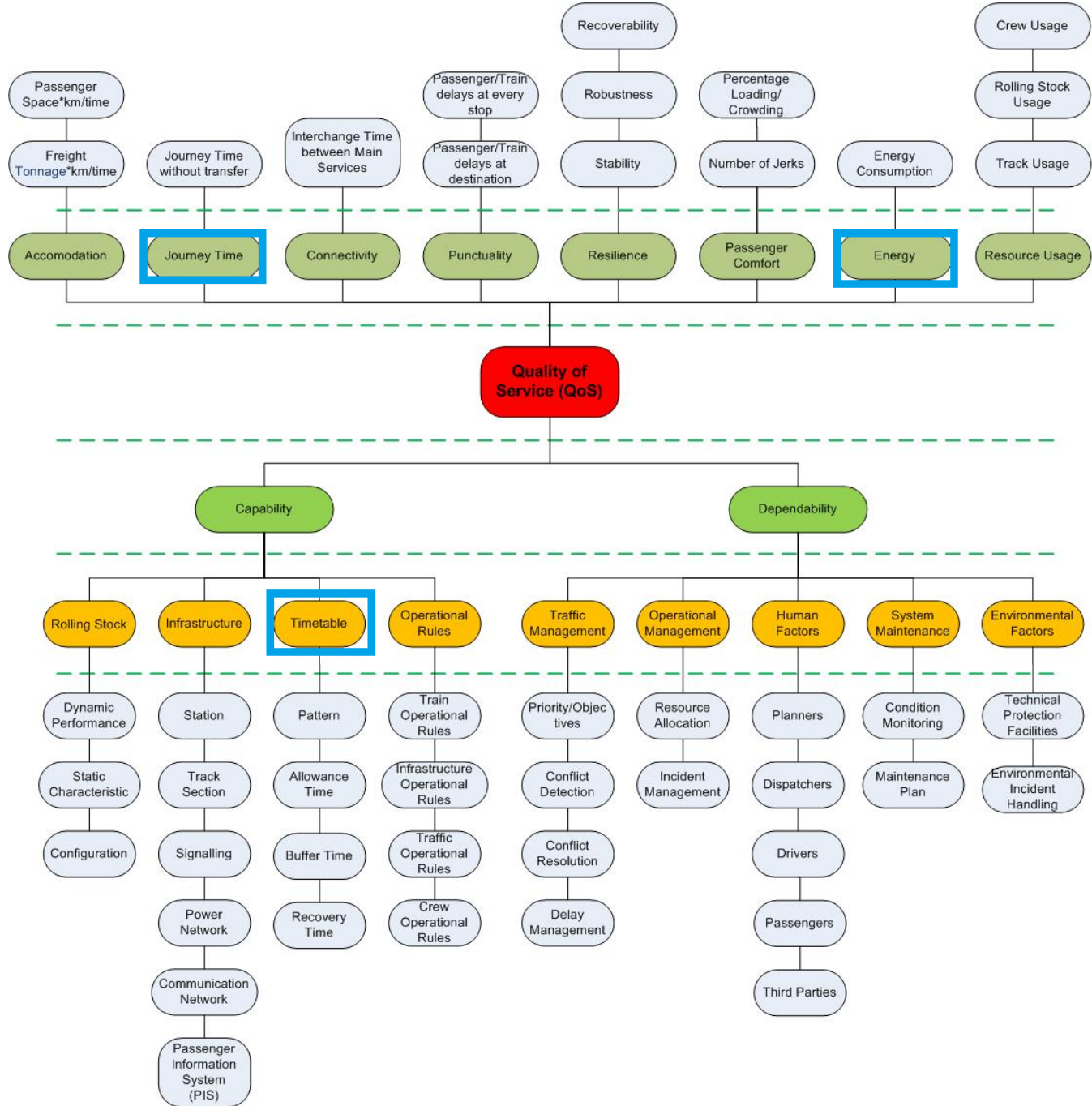


Figure 3.3: Quality of service framework (Lu et al., 2013a)

Performance indicator	Measure
Journey time (JT)	For each train with a given O-D pair, JT is defined as the time in seconds that it takes that train to travel from O to D in an ideal scenario, including dwell times at intermediate stations.
Energy (EG)	EG is defined as the total energy consumed by all the trains running on the given railway network during the time period (T) of interest in an ideal scenario.

Table 3.2: Key performance indicators

3.3 Optimisation

The cost function and constraints together with technical data on the railway network are input into the optimisation algorithm from the evaluation component. This modifies the arrival and/or the departure times of trains at one or more stations, so that the cost function is minimised (or maximised) and the constraints are met. The outcome is an optimised timetable, which enables a better quality of service on the railway network.

The algorithm that is used for the optimisation can be chosen depending on the current problem addressed by the framework. An overview of possible algorithms for solving the problem addressed in this thesis is provided in Appendix A.

As explained in the introduction to this chapter, it is not guaranteed that the optimised timetable is feasible, due to the complexity of the optimisation problem. Therefore, the timetable is automatically exported to the macroscopic stage in order to validate it and then assess the improvements brought to the quality of service on the network.

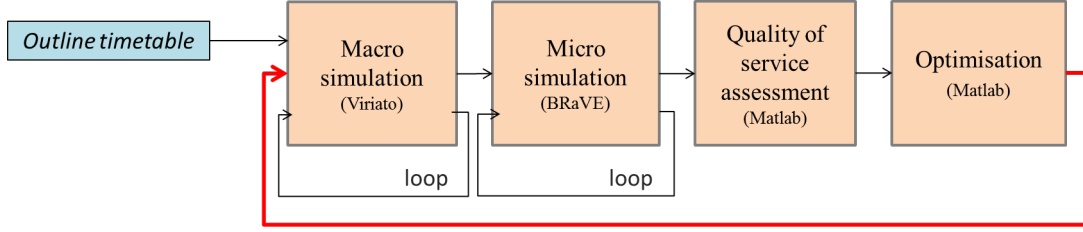


Figure 3.4: Closed-loop process

3.4 Closed-Loop Approach

The whole process, made of the four stages explained in the previous sections, is repeated for a pre-defined number of times or until the quality of service on the railway network is accepted. A red arrow in figure 3.4 emphasises the closed-loop.

A closed-loop enables the validation of a timetable and the analysis of the quality of service currently offered on the railway network under consideration.

In the next chapter a possible implementation of the framework is described.

Chapter 4

Framework Implementation

4.1 Macro Simulation

As described in chapter 3 on page 27, the first stage of the framework in figure 4.1 produces a feasible timetable using a macroscopic model of the railway infrastructure.

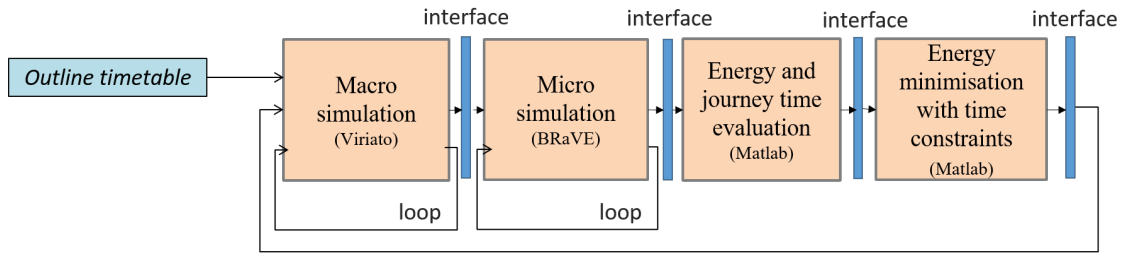


Figure 4.1: Framework software components

Automating as much of this step as possible helps timetable planners to save time in the modification of an existing timetable or in the definition of a new schedule. To this end, a tool has been chosen, Viriato, which is able to fulfil the task without being too complex, in order to limit the amount of time needed in the planning process. Section 4.1.1 introduces the reader to Viriato and its architecture.

4.1.1 Viriato

Viriato is a commercial software package that can be used by timetable planners to develop timetables according to their needs (SMA and Partner, 2009). It can be

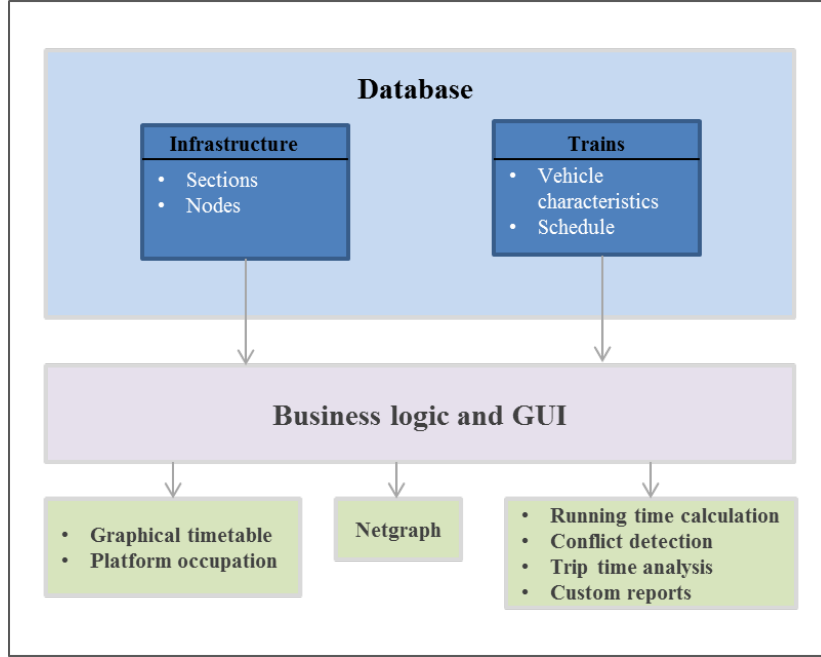


Figure 4.2: Viriato's architecture - (SMA and Partner, 2009)

employed in long- or short-term planning processes to produce feasible timetables at a macroscopic level.

Architecture

Viriato has a modular architecture: the core of the software is the business logic unit, which uses the data on infrastructure and trains stored in its database to produce graphical timetables, net-graphs, user-created reports and to perform one or more of the following functions: running time computation, conflict detection, track occupation at nodes and trip time analysis, according to the timetable planner's needs. Figure 4.2 summarises the architecture.

In this work Viriato is used to detect and solve possible unsatisfied time and infrastructure constraints of a given timetable. The non-compliance with a constraint is also referred to as a conflict. In order to perform this task, the following steps are required:

- Model definition

- Conflict detection
- Conflict resolution

Each step is described in the following sections.

Model Definition

A Viriato model is characterised by two main elements: infrastructure and trains.

Infrastructure

The infrastructure is represented at a macroscopic level, in terms of nodes and sections. A node can represent a station, a junction or a passing loop, while a section is a link between two nodes and can include intermediate nodes. The necessary input data is summarised in table 4.1. This has to be input into the software package manually. An example of infrastructure model produced by Viriato is shown in figure 4.3.

Model component	Input
Section	<ul style="list-style-type: none"> • number of tracks and direction of travel
	<ul style="list-style-type: none"> • gradient profile
	<ul style="list-style-type: none"> • speed limits
	<ul style="list-style-type: none"> • headway times
Node	<ul style="list-style-type: none"> • number of platforms with relative length
	<ul style="list-style-type: none"> • entry and exit speed limits
	<ul style="list-style-type: none"> • access limitations due to the layout
	<ul style="list-style-type: none"> • separation times between trains
Train	<ul style="list-style-type: none"> • maximum speed, gross weight and length
	<ul style="list-style-type: none"> • schedule at each node of a train route
	<ul style="list-style-type: none"> • route along a section or in a node

Table 4.1: Input to Viriato

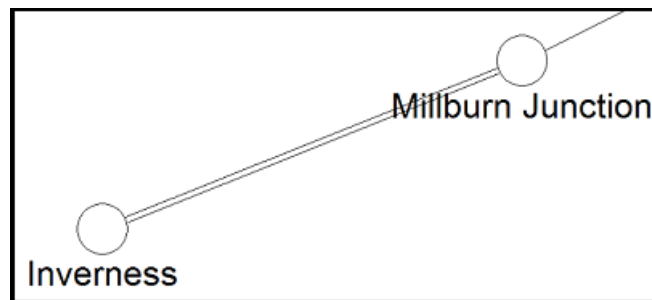


Figure 4.3: Example of a macroscopic model in Viriato

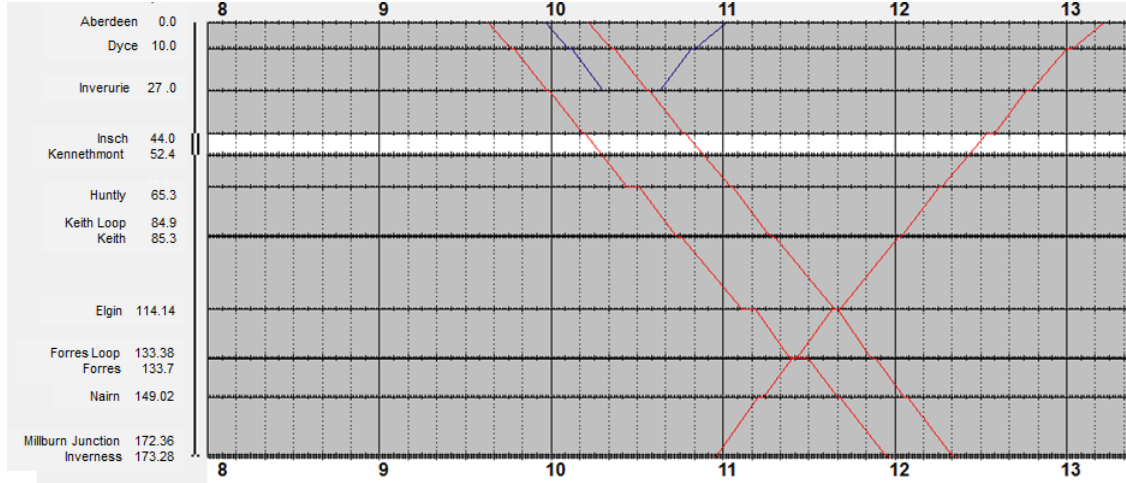


Figure 4.4: Example of timetable graph in Viriato

Trains

Trains are defined in terms of configuration, route and schedule. The input data that the timetable planner has to enter is summarised in table 4.1. An example of timetable graph is shown in figure 4.4.

Conflict Detection And Resolution

After the timetable planner has set up the model, the software package automatically detects unsatisfied constraints. The constraints regard headway times along sections, separation times at nodes, capacity and access limitations inside nodes.

The timetable planner is assigned with the task of solving conflicts between trains due to unsatisfied constraints by changing the dwell, arrival or departure time of the involved trains on part or the entirety of their route or by choosing a different platform at which a train arrives at a station.

4.2 Micro Simulation

As described in chapter 3 on page 27, the second stage of the framework in figure 4.1 on page 35 produces a feasible timetable using a detailed model of the railway infrastructure. The software solution employed to implement this step is BRaVE.

Section 4.2.1 introduces the reader to BRaVE and its role in this work.

4.2.1 BRaVE

BRaVE (Birmingham Railway Virtual Environment) is a railway operations' simulator developed by Dr Kirkwood at the University of Birmingham. It simulates the behaviour of virtual trains on the virtual track in microscopic detail. BRaVE can simulate controlled variations of dwell times, different driving styles and signalling behaviours, as well as perturbed timetable running. Its purpose in this work is to detect possible conflicts that may occur in microscopic simulation that are not detected at the macroscopic level.

In order to implement its functionality, BRaVE needs a model of the railway line and related services. Then, it can simulate train movements and issue warnings if there are conflicts between trains. The following sections describe each step in more detail.

Model Definition

The model of the railway network is automatically built by BRaVE using the Viriato model. The interface between Viriato and BRaVE is described in section 4.3 on the following page.

Conflict Detection And Resolution

The timetable planner starts the simulation of a desired Viriato model. The software simulates train movements over a period of time and issues warnings to the timetable planner if a train needs to run on a route that has already been set for another train or if a train sees an unexpected restrictive signal aspect. This means that headway times along a section and/or separation times at nodes are not met. The timetable planner has to solve possible conflicts by modifying the arrival or dwell time at stations (or passing loops) or re-routing trains at a station (or a passing loop).

4.3 Interface Between Viriato And BRaVE

Viriato stores its model in several tables of an Access database. BRaVE, which is developed using Java language, can read this format by means of Jackcess, an external open-source Java library (Health Market Science, 2013).

The timetable planner takes a snapshot of a Viriato model and feeds it into BRaVE. On the basis of the Viriato data, BRaVE automatically reproduces infrastructure, trains and timetable. The following paragraphs explain how the different model components are automatically generated by the BRaVE interface.

The interface between Viriato and BRaVE has been developed by the author of this thesis.

Infrastructure

The following infrastructure elements are automatically generated:

- nodes (stations or junctions)
- platforms inside nodes
- branches on both sides of a node
- connections between branches and platforms (or branches in the case of junctions)
- sections
- signals
- routes

The components that form the infrastructure model are listed in table 4.2 with the source used by BRaVE to draw them. Figure 4.5 shows an example of infrastructure model generated by BRaVE.

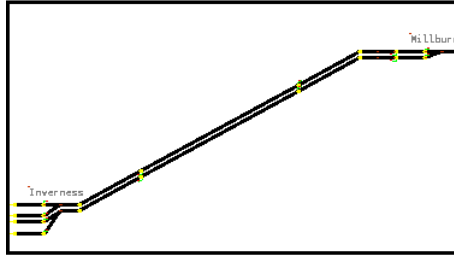


Figure 4.5: Example of infrastructure model in BRaVE

Component	Source
Node position	Coordinates of a Viriato node
Platform length and speed limits	From Viriato
Branch before or after a station	Built by the BRaVE interface
Connection between a branch and a platform (or a branch)	Built by BRaVE exploiting the Viriato data on node access restrictions due to its layout
Section length, gradient and speed limits	Built by BRaVE using Viriato data; a section is made of multiple paths, where a new path is created when either the track gradient or the track speed limit changes
Signal location	Embedded in the BRaVE interface using operational rules on the railway network under consideration
Route	Generated by BRaVE using the Viriato route as a starting point

Table 4.2: Infrastructure data in a BRaVE model

Trains

The components that characterise a BRaVE train model are listed in table 4.3 with the source used by BRaVE to build them. The train model and relative timetable is automatically generated by the BRaVE interface. An example of timetable is shown in figure 4.6.

Traffic		Timetable		Train runs		Train types		Vehicles	
Name	Train num.	Class	Train name	Train des.	Type name	Days	Start date	End date	Driver
S0	0	R	SC195 R 45	Aberdee...	SC195 R 45 MTWT...	2013-12...	2014-05...	SilviaDri...	
S1	0	E	SC195 E 29	Aberdee...	SC195 E 29 MTWT...	2013-12...	2014-05...	SilviaDri...	
S2	0	E	SC195 E 41.1	Aberdee...	SC195 E 41.1 MTWT...	2013-12...	2014-05...	SilviaDri...	
S3	0	R	SC195 R 24	Invernes...	SC195 R 24 MTWT...	2013-12...	2014-05...	SilviaDri...	
S4	0	E	SC195 E 50	Invernes...	SC195 E 50 MTWT...	2013-12...	2014-05...	SilviaDri...	
Route	Departure	Min stop	Stop	Running ti.	Req. Arrival	Req. Depa.	Type	Track ID	Station ID
R03	09:17:00	0	0	0	09:17:00	09:18:00	stop	N28	ARD
R115	09:47:00	60	0	0	09:47:00	09:47:00	stop	N58	DYC
R92	09:59:30	60	0	0	09:59:30	09:59:30	stop	N25	DNK
R84	10:12:00	60	0	0	10:12:00	10:12:00	stop	N14	JNS
R105	10:17:42	0	0	0	10:17:42	10:17:42	pass	N45	KNM
R109	10:31:00	270	0	0	10:31:00	10:31:00	stop	N50	HNT
R98	10:43:30	0	0	0	10:43:30	10:43:30	pass	N33	KEH LP
R86	10:45:30	60	0	0	10:45:30	10:45:30	stop	N17	KEH
R103	11:11:30	300	0	0	11:11:30	11:11:30	stop	N40	ELG
R82	11:28:30	330	0	0	11:28:30	11:28:30	stop	N9	FOR LP
R88	11:30:00	30	0	0	11:30:00	11:30:00	stop	N20	FOR
R79	11:41:00	60	0	0	11:41:00	11:41:00	stop	N2	NRN
R111	11:56:30	0	0	0	11:56:30	11:56:30	pass	N53	MB JN
R121	11:58:30	0	0	0	11:58:30	11:58:30	stop	N65	INV

Figure 4.6: Example of timetable in BRaVE

Input	Source
Train maximum speed, gross weight and total length	From Viriato
Train tractive effort curve	Generated by BRaVE using the train data from Viriato
Train resistance curve	Embedded in BRaVE
Train maximum acceleration and deceleration	Embedded in BRaVE based on the train characteristics
Number of vehicles	Calculated based on the train length or using a default value depending on the train type
Schedule	From Viriato
Route	From Viriato

Table 4.3: Train data in a BRaVE model

4.4 Viriato And BRaVE Comparison

Comparing the input that is required by a Viriato model (table 4.1 on page 38) with the one of a BRaVE model (table 4.2 on the preceding page and table 4.3), it is clear that building a Viriato model is less time-consuming and does not require

as accurate knowledge of a given railway system as a BRaVE model. In order to build a model in BRaVE, signals, points and train detection systems need to be included. Moreover, the train model requires data on the traction system and the train configuration in order to reproduce the train dynamics. Finally, a train route is represented at the level of microscopic nodes, whereas it is formed of macroscopic nodes in Viriato. Generating a microscopic model of a railway network automatically allows timetable planners to save time.

Considering the simulation platform, Viriato offers a more user-friendly support for manual conflict resolution. The timetable over a desired time period is displayed graphically and conflicts can be solved by selecting a train and entering the amount of time deviation to a train journey.

Moreover, Viriato allows to visualise all the conflicts at the same time. In BRaVE, instead, the microscopic simulation does not show a conflict until this occurs. Consequently, it is not possible to have a global view of the conflicts. It is therefore more difficult to foresee the impact of a modified train journey on the other train journeys.

Solving conflicts at a macroscopic level first reduces the complexity of the planning process and potentially speeds up the construction of a timetable.

4.5 Timetable Assessment

As explained in chapter 3 on page 27, the evaluation stage assesses the feasible timetable outputted by the microscopic simulator in terms of the total energy consumption in a given time period. The total journey time per train is also shown to analyse how this varies over iterations. The architecture for evaluating a timetable was developed using Matlab by Dr G.L. Nicholson at the University of Birmingham (Nicholson et al., 2015). In more detail, the evaluation module obtains data on the actual train schedule and its energy consumption via a log file produced by BRaVE during the simulation and displays the information using matrices and graphs. Figure 4.7 shows an extract from a simulation, where actual and scheduled arrival times at each node together with the cumulative energy consumption are highlighted by a red rectangle.

A	S2	S2_1	N28	09:38:00	09:38:00	0	0	0
D	S2	S2_1	N28	09:38:00	09:38:00	0	0	0
A	S2	S2_1	N58	09:47:01	09:46:00	29.95816	9770	0
D	S2	S2_1	N58	09:48:01	09:47:00	29.95816	9770	0
A	S0	S0_1	N30	09:58:00	09:58:00	0	0	0
D	S0	S0_1	N30	09:58:00	09:58:00	0	0	0

Figure 4.7: BRaVE - Simulation data

Two Matlab matrices store the train actual arrival and departure times at each node; rows represent trains, columns contain node ids in alphabetical order and each cell stores the actual arrival or departure time at the node (in seconds), if the train route includes that node, a default value, otherwise. An example of a matrix containing train departure times is shown in figure 4.8. The two matrices are then combined by the software to form a three-dimensional matrix that, for each train, contains the journey time between each pair of nodes that are part of a train route, a default value, otherwise. An example of a journey time matrix for one train is displayed in figure 4.9.

Node id Train id	Node id					
	ABD	DYC	ELG	FOR	...	NRN
S1	35880	36428	NaN	NaN	...	NaN
S2	36780	37283	42139	42971	...	43634
S3
S4
S5	47669	46988	42073	41212	...	40423

Figure 4.8: Evaluation - Departure time matrix

Service S1						
Node id Node id	Node id					
	ABD	DYC	ELG	FOR	...	NRN
ABD	0	548	NaN	NaN	...	NaN
DYC	NaN	NaN	NaN	NaN	...	NaN
ELG	NaN	NaN	NaN	NaN	NaN	NaN
FOR	NaN	NaN	NaN	NaN	NaN	NaN
...
NRN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 4.9: Evaluation - Journey time matrix for service S1

The same procedure is used by the program to create a three-dimensional matrix that stores the values of energy consumption.

A graphical representation of the selected indicators is also available to timetable planners. The total journey time of each train can be visualised in one graph, while the global energy consumption on the network in a given time period is displayed in a separate graph. Figure 4.10 shows an example of the two graphs. Figure 4.10a represents the total journey time per train, in which each train is identified by an id (S1, ..., S5). Figure 4.10b shows instead the total energy consumption on the given railway network over a time period at each iteration of the proposed process.

The program is used by timetable planners to select the indicators that form the cost function and to define the constraints that trains have to respect. An optimisation problem is therefore formulated.

Cost function, constraints and technical data on the railway system (such as number of trains, number of sections forming a train route, speed limits and node

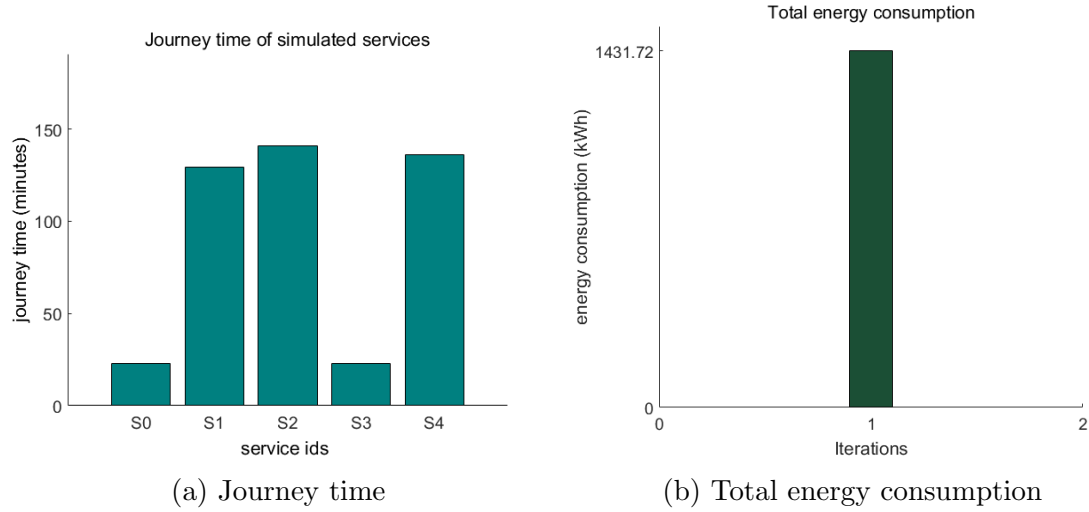


Figure 4.10: Evaluation step - Graphs

locations) are automatically transferred from the evaluation stage into the optimisation algorithm. The Matlab interface is described in the following section.

4.6 Interface Between Evaluation And Optimisation Stages

The data is transferred between the evaluation and the optimisation stages via an XML file. XML (Extensible Markup Language) was chosen as a format for data exchange since it is an accepted standard developed by the W3C consortium (W3C, 1994) and Matlab offers good support for it. BRaVE generates the part of the XML file relative to the technical data on the given railway network. The file exchange is explained in the following sections.

BRaVE - Writing The Data In An XML File

The data of an XML file is arranged in a tree structure, which is made of a root element and branches elements that are further split until no more branches are generated (the leaves). Figure 4.11 on the following page shows an example of XML file, in which the root element is identified by the tag `<allTrains>` and contains all the elements that refer to single trains, tag `<TrainData>`. The data relative to each

train is stored in corresponding elements in the form of text. For example, a train name is kept as a text node inside the element `<trainName>`.

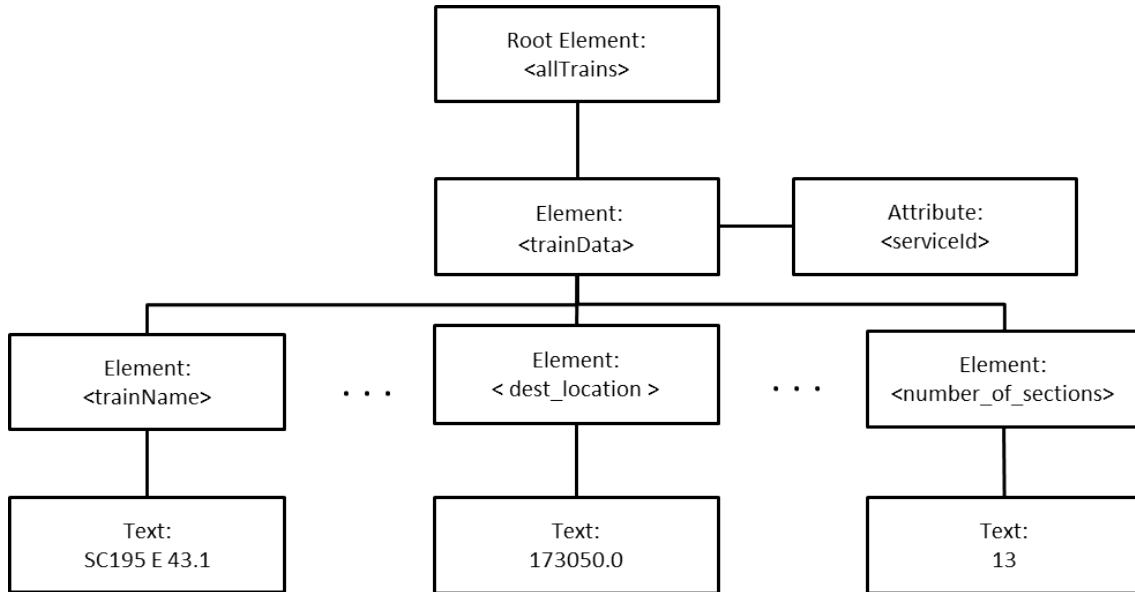


Figure 4.11: Example of XML structure

In order to write the file, the properties of Java classes are serialised into an XML stream. This means that if, for example, there exists a Java class named `TrainData` with a property that represents the train name, the latter will be converted by BRaVE into an XML element with tag `<trainName>`. Java, the language in which BRaVE is developed, provides a framework for serialising Java classes into an XML-structured file: JABX, Java Architecture for XML Binding. Information on the framework can be found in (Oracle, 2003).

Matlab - Importing The Content Of An XML File

The content of the XML file is read by Matlab and stored in a structure, in which each variable groups the content of the same XML element for different trains. For example, the vector *destination_loc* is formed of n cells, one per train, containing a single-train destination's location. Matlab has built-in functions for reading an XML file. The program extracts each element from the XML file and stores it in the pre-defined structure.

4.7 Optimisation Algorithm

In this implementation, the trains that form a timetable are optimised sequentially. The train inter-node cruising speeds are the variables of each sub-problem, where a node can be a station, a passing loop or a junction. Either a brute force or a genetic algorithm is used to optimise the train cruising speeds depending on the number of variables that constitute the problem. The algorithms have been developed using Matlab and are described in the following.

4.7.1 Brute Force Algorithm

The brute force algorithm generates all the possible combinations of cruising speeds. Then, it tests the feasibility of each combination by calling BRaVE to simulate the timetable with the current set of speeds and discards infeasible combinations. Finally, it ranks the solutions in ascending order of total energy consumption and returns the one with lowest value as output.

4.7.2 Genetic Algorithm

Genetic algorithms were developed by Holland and his team in the 1960s (Holland, 1992). They imitate the biological evolution process, in which a population of individuals evolves by means of crossover and mutation. As explained in (Michalewicz, 1996), individuals or chromosomes are made of units, genes, each of which controls one or more features in an individual. In the optimisation context, an individual represents a solution to a given problem. The evolution of a population of individuals corresponds to a search in the space of potential solutions. The basic structure of a genetic algorithm is made of several iterations, in each of which each individual is evaluated in terms of fitness using a defined measure and the fittest individuals are selected to generate the new population using genetic operators. The basic operators are crossover and mutation. Crossover combines the genes of two individuals,

also called parents, to form two offspring or children by for example swapping segments of the parents. In this way genetic features are exchanged. Mutation instead changes one or more genes of selected individuals with a probability equal to the mutation rate. Variability into the population is thus introduced.

The structure of the genetic algorithm developed by the author is shown in figure 4.12. The following paragraphs describe each step.

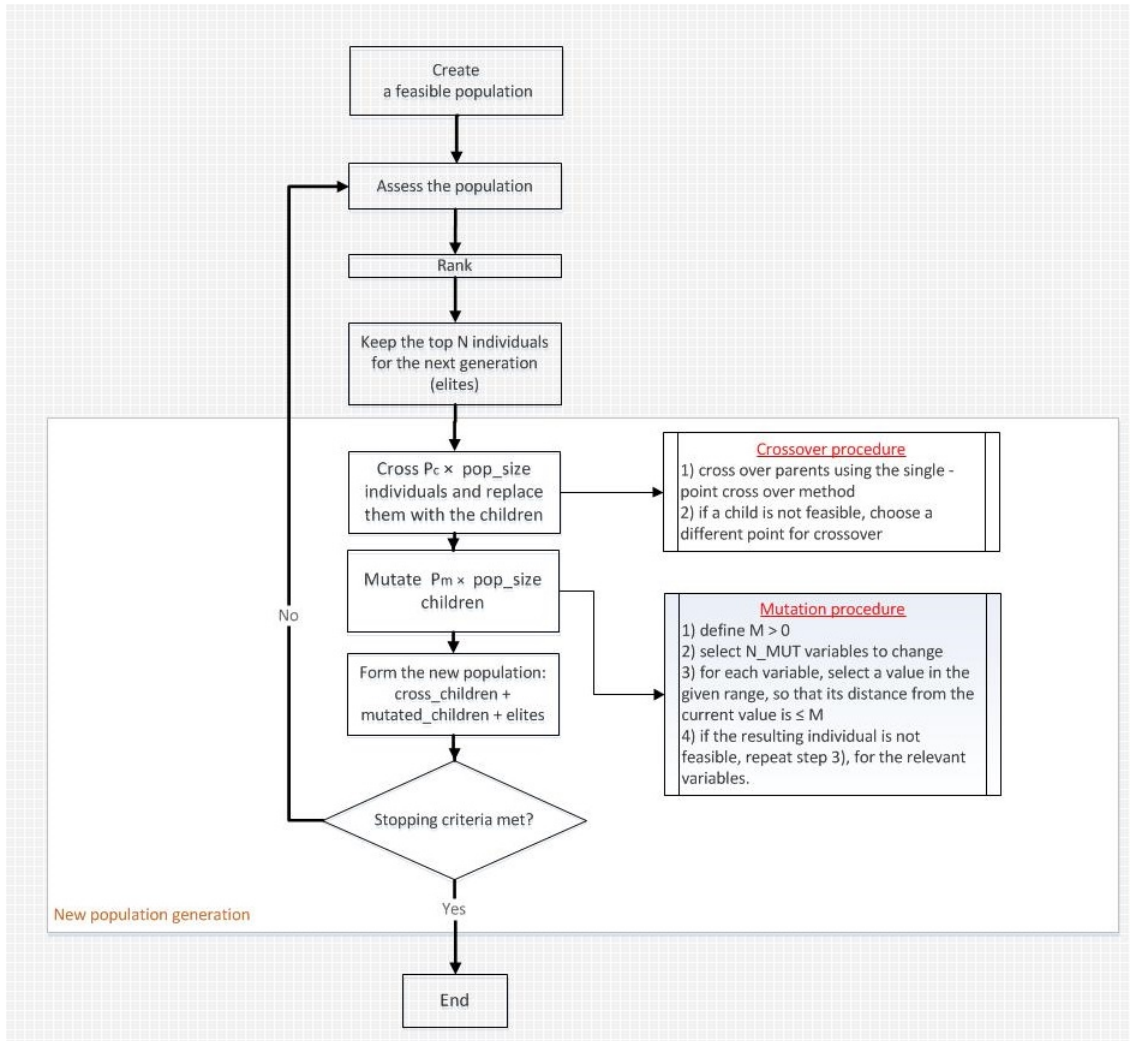


Figure 4.12: Genetic algorithm implementation

Initialisation

The initial population is created using a first solution to define the variable bounds; in more detail, two cases have to be distinguished: in the first case, a train does

not have time constraints at specific locations; therefore, its bounds are not changed and individuals are created by randomly selecting speeds within the bounds. In the second case, a train has time constraints to meet. A first simulation is performed in which the train runs at maximum speed and its time constraints are checked. If the train fulfils all the constraints, the speed set is used to refine the variable lower and upper bounds, so that the lower bound (upper bound) is below (above) the determined speed by a given step size, otherwise a subset of the speeds is selected and changed; the process is repeated until a feasible speed set is obtained. Individuals are then generated by randomly selecting section speed values in the defined bounds. In order to test the feasibility of a given speed set, the BRaVE simulator is called by the program. The procedure is summarised in figure 4.13, in which it is assumed that a train interacts with only one train.

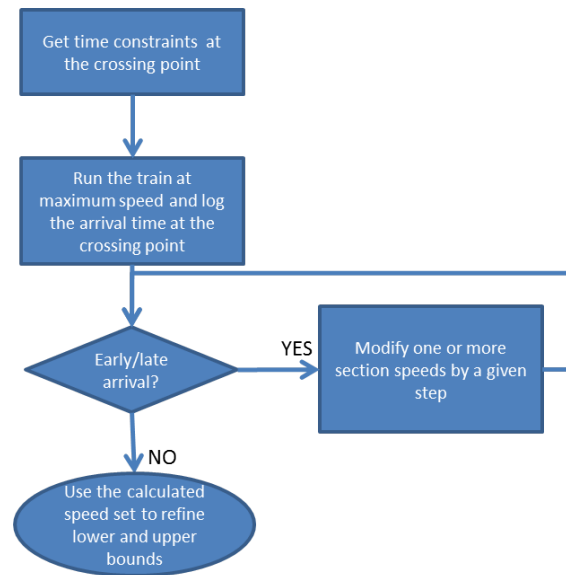


Figure 4.13: Genetic algorithm - Initialisation procedure

Assessment

As a second step, the formed population is assessed and ranked in ascending order of score, the fitness, which is based on the train energy consumption from origin to destination stations, as shown in equation (4.1):

$$f(x) = \min \sum_{i=1}^n eg_i, \quad (4.1)$$

where eg_i is the energy on a single section of a train's route. The score is used to calculate a ranked score based on the position of the individual in the ordered population (equation 4.2). This avoids the effect of the spread of the raw scores.

$$ranked_score = \frac{1}{\sqrt{i}}. \quad (4.2)$$

Crossover


A roulette-wheel strategy is used to choose the parents. A roulette-wheel is divided into slices, the size of each of which depends on the cumulative probability of an individual. The latter is related to the individual's fitness or score. Individuals are then selected by spinning the wheel.

Using this strategy, cumulative scores, $score_{cum}$, are obtained from the ranked ones and the parents are chosen by generating random numbers in the range

$[\min(score_{cum}), \max(score_{cum})]$. The same individual can be selected multiple times, with a probability that depends on the relative cumulative score. Given a crossover probability P_c , $P_c \times Pop_size$ individuals are crossed over to create $P_c \times Pop_size$ children that will be part of the new population. A single-point crossover method is used for crossing the parents; this means that, for each pair of parents, a random number p in the range $[2, num_{vars} - 1]$ is generated and used as a location for splitting each parent vector into two parts. Two children are created, whose vector entries will have the values of one parent up to the point p and the values of the other parent from p to the end of the vector. The crossover point cannot be the first or the last entry of the parents' vector in order to avoid the generation of children that are copies of the parents. The procedure is illustrated in figure 4.14 on the following page. If one of the two children is not feasible, the

feasible child is kept and the operation is repeated to get a second feasible child, using another crossover point and the same pair of parents.

	p												
Parent 1	98	95	95	96	96	112	64	88	120	32	108	94	17
Parent 2	96	95	97	96	96	112	64	87	120	32	108	95	17



Child 1	96	95	97	96	96	112	64	87	120	32	108	94	17
Child 2	98	95	95	96	96	112	64	88	120	32	108	95	17

Figure 4.14: Example of application of the single-point crossover method

Mutation

The children generated from crossover form the set from which, given a mutation probability P_m , $P_m \times pop_size$ individuals are selected for mutation and used to replace the current ones. For each individual, N_MUT variables are randomly chosen, one at a time, and their value is replaced with a random one that is included in the variable bounds and is at a maximum distance M from the current value. If the resulting individual is not feasible, the N_MUT variables that caused the infeasibility are assigned a different value and the individual is tested again. The process is iterated until $P_m \times pop_size$ feasible individuals are obtained. An example of how the mutation operator works is shown in figure 4.15, where v_1 and v_2 are the vector entries that are changed.

	<div><div>v_1</div><div>v_2</div></div>												
Parent	98	95	95	96	96	112	64	88	120	32	108	94	17
	<div>↓</div>												
Child	98	95	95	96	96	112	64	88	116	32	108	97	17

Figure 4.15: Mutation operator

Process Iteration

The new population is formed of the individuals generated by crossover and mutation and of the best ($pop_size - crossover$) individuals, the elites, which are passed to the next generation without being manipulated. The assessment, crossover and mutation steps are repeated until one of the defined stopping criteria is met. In this implementation, three different criteria are considered:

1. maximum number of iterations
2. average change of the fitness function over the last fifty generations less than or equal to a given threshold
3. maximum computational time since the algorithm started

If one of the above criteria is fulfilled, the algorithm ends and the current solution is returned.

Chapter 5

Case study: Aberdeen-Inverness Railway Timetable Design

The proposed framework has been used to develop a higher-capacity and energy-efficient timetable on the Aberdeen-Inverness railway line when a small subset of trains is taken into account.

This line is single track with either a token or a tokenless block system on each inter-node section and passing loops. Therefore, only one train at a time can occupy the same section track. A single-track line has been chosen because it forms a more interesting case than a multiple-track line in terms of train interactions. The line in question has a variable gradient and speed profile.

A high level map of the line is shown in figure 5.1, whereas the infrastructure layout of a short stretch of the line is displayed in figure 5.2 on the next page.

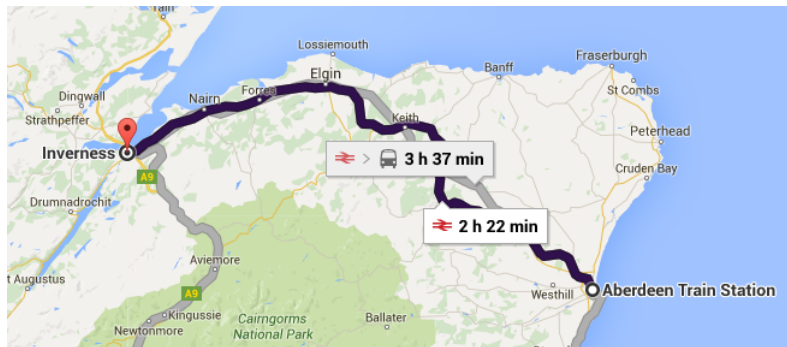


Figure 5.1: Case study - Aberdeen-Inverness railway line

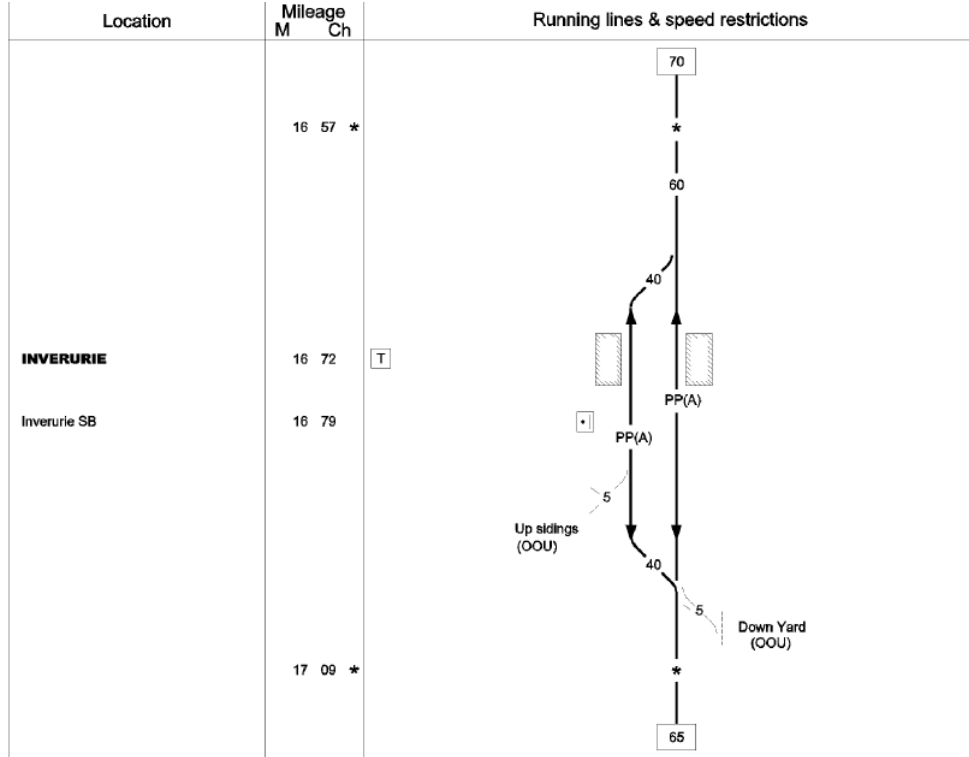


Figure 5.2: Case study - Infrastructure layout of a stretch of the Aberdeen-Inverness railway line (Network Rail, 2014)

The Winter timetable 2013 is used as a starting point for this work. A subset of four trains is selected from the timetable and one extra train is added to it. The selected trains have different origin and destination stations along the line. Two types of train are involved; their technical characteristics are shown in table 5.1.

Class	Mass (tonnes)	Power (kW)	Max accelera- tion rate (m/s^2)	Max decelera- tion rate (m/s^2)
158	88.00	520	0.8	0.7
170	147.74	945	0.8	0.7

Table 5.1: Train technical data

The framework is used to modify the timetable so that more energy-efficient operations are enabled on the railway line.

The following sections explain how the process is implemented for the case study and discuss the obtained numerical results.

5.1 Process Deployment

A draft timetable for the Aberdeen-Inverness railway line is obtained from the existing Winter 2013 timetable.

The draft timetable is made of two trains travelling towards Inverness and two trains heading towards Aberdeen. One extra train is added to the timetable.

The outline timetable and the technical data on the infrastructure obtained from documents provided by Network Rail (2014) are used by the timetable planner to build the Viriato macroscopic model of the railway system under consideration as explained in the following section.

5.1.1 Viriato Model Definition

A Viriato model requires input data on infrastructure and trains; a list of the required data was provided in section 3.1 on page 28. In terms of infrastructure, sections and nodes need to be defined. Length and number of tracks of a section can be input via the graphical interface shown in figure 5.3 on the next page. Using the same panel, headway times can be input. The speed limits imposed on trains travelling along a section and the gradient profile of that section are input via the interfaces displayed in figure 5.4. A speed limit or a gradient value with corresponding validity distance need to be entered into the software.

Regarding nodes, number of platforms and relative length can be input via the screen shown in figure 5.5 on page 59. Entry/exit speed limits to each platform have to be specified together with the distance they start or finish at (figure 5.6 on page 59). Route limitations inside a station and separation times between trains due to the station layout can be input into the program via the interfaces displayed in figure 5.7 on page 60.

Section Definition

Section ID: SC195_1 Description: Dyce - Aberdeen Valid from: Valid until: Status: Last Change: 2014-03-27 14:56 Remarks:

☐ Attributes ☒ Tracks ☐ Headway Times

Node ID	Node Name	Km 1	Km 2	# Tracks	Km from Start	Interm. Distance	Direction	1	Valid from	Valid until	Status
DYC	Dyce	10.000			0.000		B				
ABD	Aberdeen	0.000		1	10.000	10.000	B	1	B		

Node: Ins. Above Remove Ins. Below Search

Section: New Copy Rename Delete Print Save Close

Figure 5.3: Viriato - Definition of the Aberdeen-Dyce section

Infrastructure Data for Running Time Calculator

Section ID: SC195_1 Description: Dyce - Aberdeen Valid from: Valid until: Status:

(a) Speed Limits

Speed profile type: 1 - SC100 Direction: A - Ascending

Km 1	Km 2	Distance from Start	Intermediate Distance	Speed Limit	Node Name
10.000		0.000			Dyce
10.000		0.000	0.580	88	
9.420		0.580	2.860	104	
6.560		3.440	0.040	88	
6.520		3.480	3.140	104	
3.380		6.620	0.580	88	
2.800		7.200	2.800	64	
0.000		10.000			Aberdeen

Insert Copy from Reverse Delete Print Save Close

(b) Gradient profile

Km 1	Km 2	Distance from Start	Intermediate Distance	Profile [1/1000]	Node Name	Height above sea level (m)
10.000		0.000			Dyce	
9.660		0.340	0.316	-5.1		50.3
9.344		0.656	0.529	0.0		48.7
8.915		1.195	0.495	-10.1		48.7
8.320		1.690	0.593	-7.1		43.7
7.727		2.273	0.255	0.0		39.5
7.472		2.528	0.448	-5.6		39.5
7.024		2.976	2.049	-3.6		37.0
4.975		5.025	0.111	0.0		29.6
4.864		5.136	0.864	1.6		29.6
4.000		6.000	0.080	0.0		31.0
3.920		6.080	0.485	-6.2		31.0
3.455		6.545	0.223	0.0		28.1
...	

Insert Delete ☒ Define gradient by height above sea level Print Save Close

(a) Speed limits on the Aberdeen-Dyce section of the railway line (b) Gradient profile on the Aberdeen-Dyce section of the railway line

Figure 5.4: Viriato - Section model construction

Finally, configuration, route and schedule of trains are input via the graphical panel shown in figure 5.8 on page 60. This data is retrieved from train technical data sheets and the Working Timetable 2013.

Platform ID	Track		Platform Length	Stopping place	
	Name	Length		Position [km]	Alignment
2	2	128	128	44.040	<- >
1	1	130	130	44.040	<- >

Figure 5.5: Viriato - Definition of the number of platforms and corresponding length at Insch station (Network Rail, 2014)

Speed limits for:
☒ Left Station Head
☐ Right Station Head

Reference section: SC195_4: < 44.040 km

Definition of specific start and end points: ☐

Entry speed limits [km/h]

from	to	Start [km]
SC195_4-2	96	44.040
End [km]	44.040	44.040

Exit speed limits [km/h]

to	from	End [km]
SC195_4-1	64	44.040
Start [km]	44.040	44.040

Figure 5.6: Viriato - Definition of entry/exit speed limits at Insch station (Network Rail, 2014)

5.1.2 Viriato Conflict Detection And Resolution

The obtained timetable can be visualised by means of a graphical panel, as in figure 5.9. Timetable planners can check the feasibility of the timetable using the conflict detection function displayed on the timetable graph in figure 5.10 on page 61. Possible conflicts are marked by the software with red strips. An example is shown in figure 5.11 on page 62.

The timetable planner is assigned the task of solving the conflicts. Two actions need to be performed:

- decide which of the trains involved should be delayed;

Station and Junction Definition

☒ Stations NodeID: INS Node Name: Insch

☐ Junctions

Incompatible routes Separation times Entry/exit speeds **Itinerary limitations**

Section track	Station track	Entry Impossible	Exit Impossible
SC195_4-2	2		
	1	X	
SC195_4-1	2		X
	1		
SC195_3-1	2	X	
	1		X

Station and Junction Definition

☒ Stations NodeID: INS Node Name: Insch

☐ Junctions

Station tracks Section tracks **Separation**

Incompatible routes

Basic separation times [min]

Departure/arrival (same station track) 0

Arrival/departure (same section track) 1.5

Arrival/passage (same section track) 4.5

☒ Simultaneous arrivals impossible

Arrival/arrival at the station 1

(a) Route restrictions inside Insch station (b) Separation times between trains at Insch station (Network Rail, 2014)

Figure 5.7: Viriato - Node model construction

SC195 E 29 / 2014 sxu256 19/09/2014 11:15:00

Train ID: SC195 E 29 Description: Aberdeen - Inverness Notes:

Train Group: Long Distance Service Version ID: 2014 ☒ Active

Train Numbers (1) **Schedule (2)** Filters (3) Train Basic Data (4)

Operating Day/Validity: 15 First Train: 10:13.0 Fixed Node: Aberdeen ☐ Display nodes with ID

Time Interval: Single train Last Train: 10:13.0 Timetable Period: w12014 ☐ Enter Departure Times

Source of Running Time: v (km/h): 81

Section	Sect. Track	Node	Track No.	Track Info	ST	Arrival Time	Dep. Time	Run Time	Add Run	Sup Run	Stop Time	Add Stop	Sup Stop
		Aberdeen	2				10:13.0						
SC195_1	1	Dyce				10:21.0	10:22.0	8.0			1.0		
SC195_2	1	Inverurie				10:33.5	10:34.5	11.5			1.0		
SC195_3	1	Insch			x	10:46.0	10:47.0	11.5			1.0		
SC195_4	1	Huntly				11:02.5	11:03.5	15.5			1.0		
SC195_5	1	Keith Loop					11:16.0	12.5					
SC195_5	1	Keith				11:17.0	11:18.0	1.0			1.0		
SC195_6	1	Elgin	2		x	11:39.0	11:40.0	21.0			1.0		
SC195_7	1	Forres Loop					11:51.5	11.5					
SC195_7	1	Forres				11:52.5	11:53.5	1.0			1.0		
SC195_8	1	Nairn				12:03.5	12:04.5	10.0			1.0		
SC195_9	1	Millburn Junction					12:20.0	15.5					
SC195_9	2	Inverness	4			12:22.0		2.0					

Figure 5.8: Viriato - Train route and schedule definition

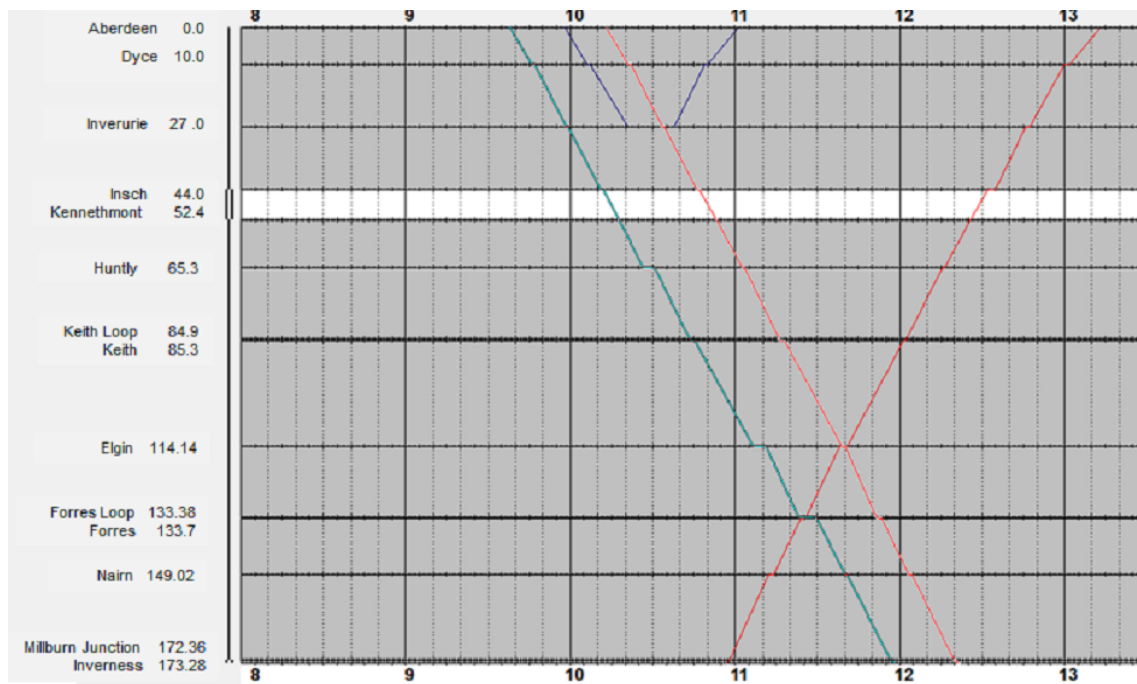


Figure 5.9: Viriato - Timetable with the additional train

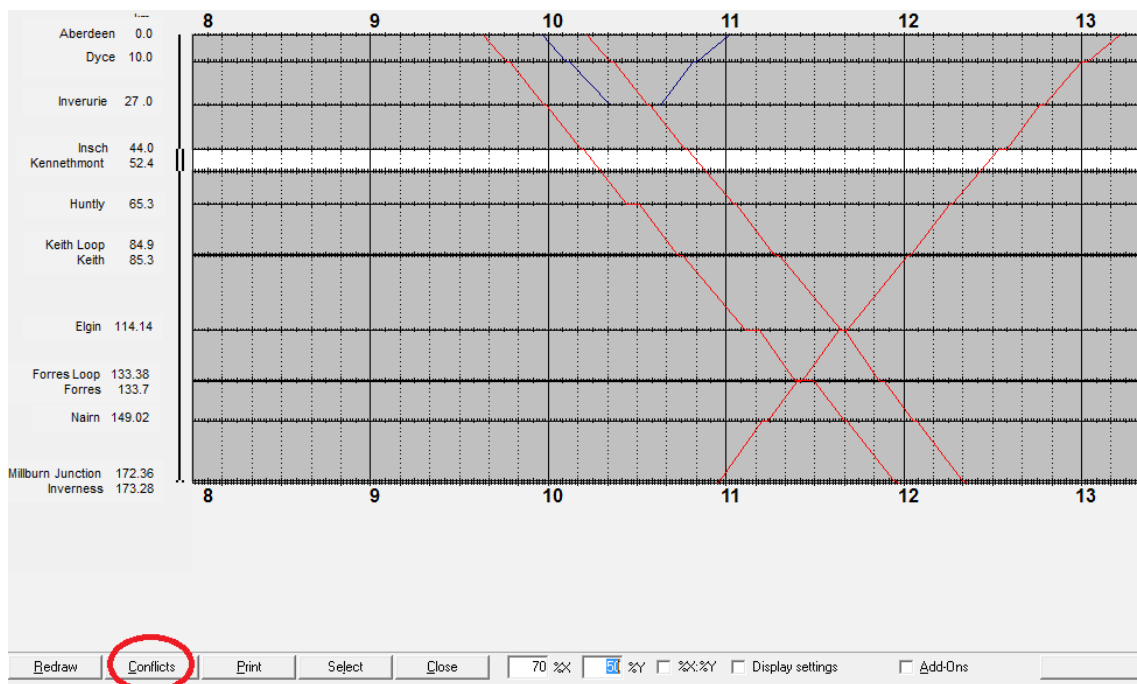


Figure 5.10: Viriato - Conflict detection

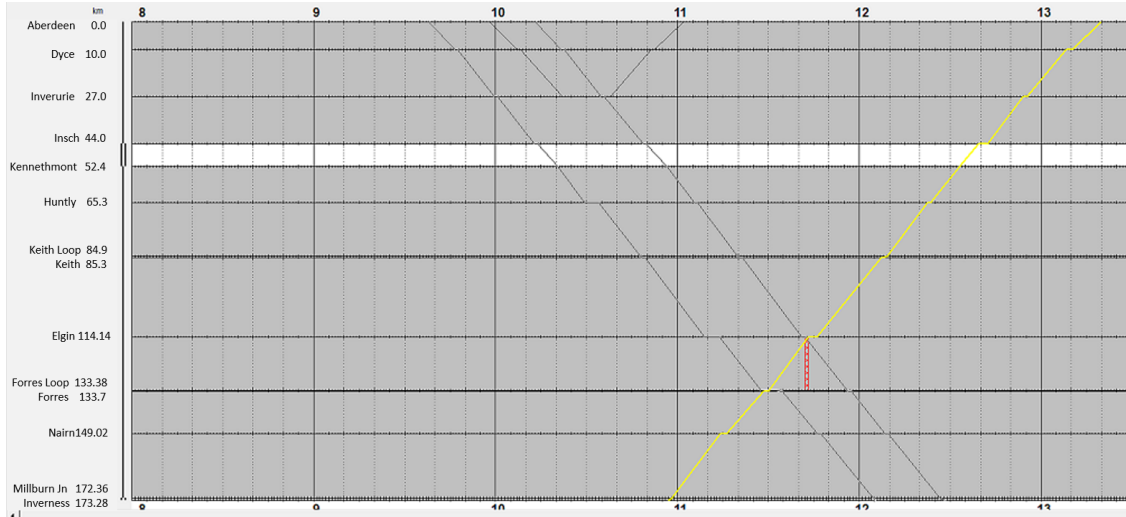


Figure 5.11: Viriato - Timetable after the first optimisation

- modify the arrival and, possibly, the departure times of the selected trains at the relevant stations.

Viriato supports the timetable planner in the manual conflict resolution by offering a global view of all the conflicts in the selected time period and by promptly displaying the amended timetable on the screen.

Once no conflicts are detected by the software, the timetable can be automatically imported into BRaVE. This process is described in the following section.

5.1.3 BRaVE Model Definition

The timetable planner has to select the Viriato model they want to import, as displayed in figure 5.12 on the following page. The model is automatically built by BRaVE and displayed on the screen (figure 5.13).

5.1.4 BRaVE Conflict Detection And Resolution

The timetable planner can then start the simulation and if, for example, a train is unexpectedly held at a red signal, a warning is issued. This means that the timetable is not well designed. The track section currently occupied by the train is highlighted in red. The process is shown in figures 5.14 and 5.15.

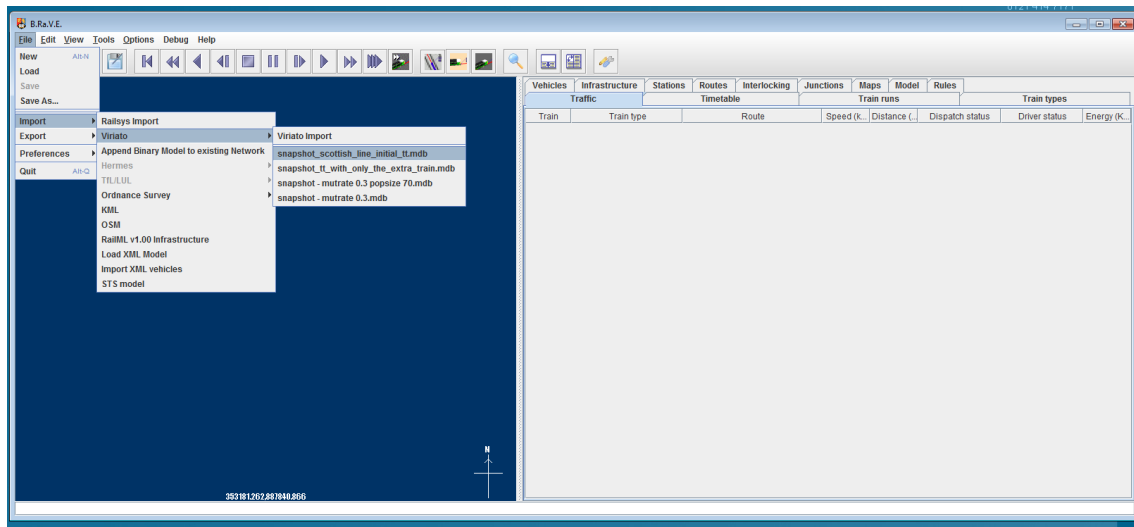


Figure 5.12: Import of a Viriato model into B.Ra.V.E

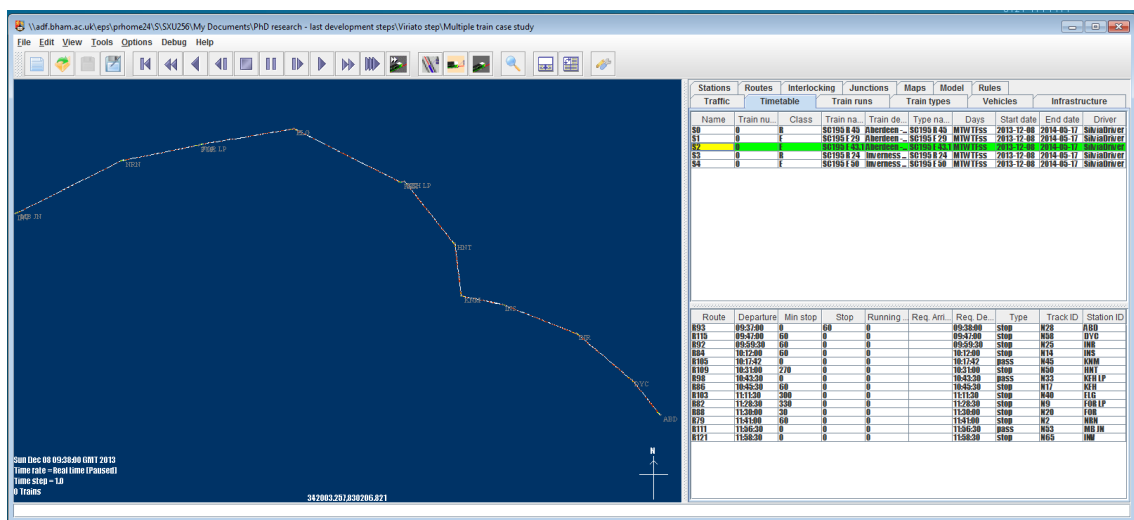


Figure 5.13: B.Ra.V.E model

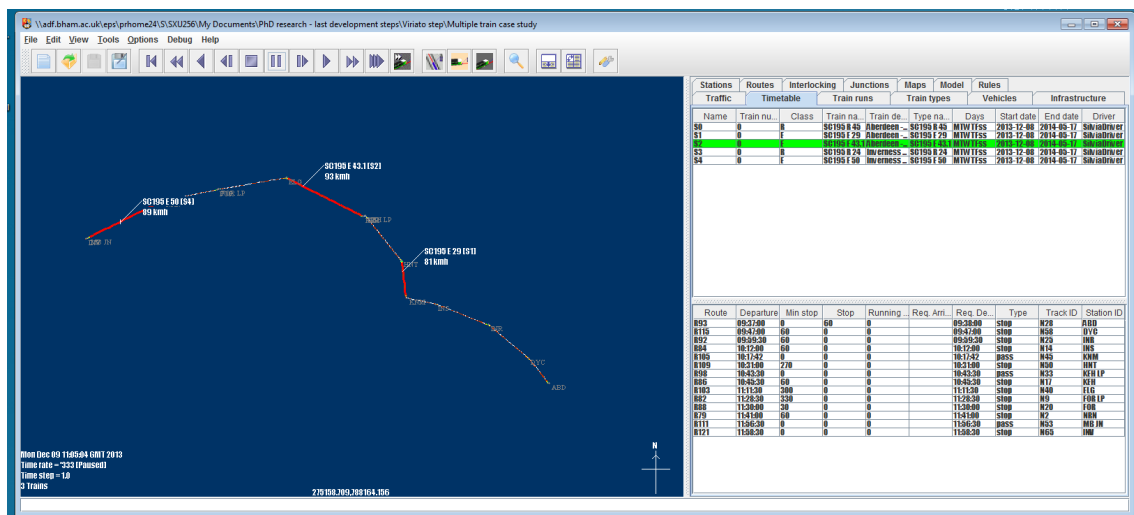


Figure 5.14: BRaVE simulation

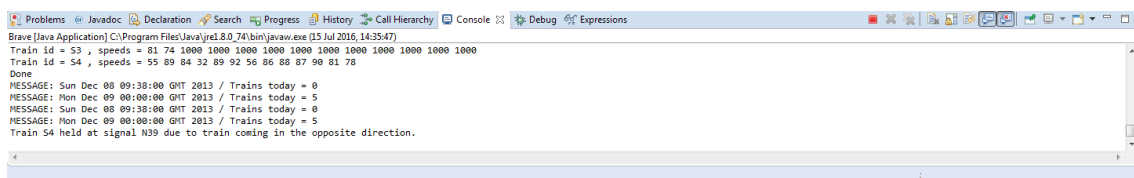


Figure 5.15: BRaVE - Example of warning issued

The timetable planner is again asked to solve the conflicts manually. They carry out the same steps explained for the conflict resolution process in Viriato. The BRaVE simulator will display one warning at a time, when a conflict occurs. It becomes clear that understanding and predicting the impact of a decision on the trains of the network is harder than in Viriato. Using Viriato brings benefits to the amount of time spent by timetable planners in setting up the model of the railway network and in solving possible conflicts. The BRaVE simulator is however necessary for assuring conflict-free train movements.

5.1.5 BRaVE Logging Functionality

Before starting the simulation, the timetable planner has to start the logging functionality, as displayed in figure 5.16 on the next page.

The actual train arrival time and cumulative energy consumption at relevant nodes on each train route are recorded during the simulation time and stored in

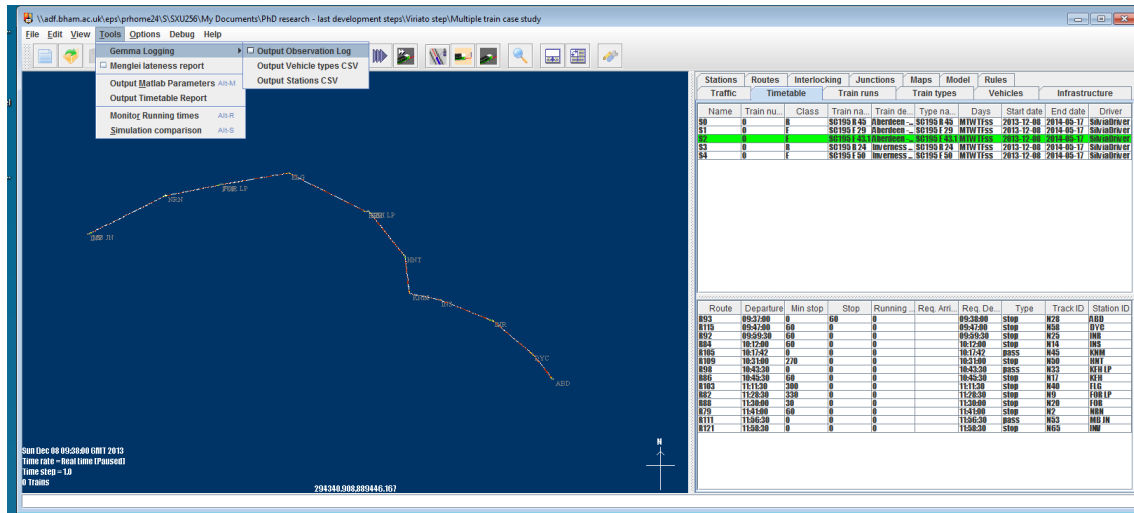


Figure 5.16: BRaVE - Logging simulation data

a csv file. Figure 5.17 on the following page shows an extract from the log file produced when the outline timetable is simulated.

5.1.6 Quality Of Service Assessment

The timetable planner starts the Matlab evaluation process by selecting the file containing the main program (figure 5.18 on page 67).

The simulation data is automatically combined by the software and two matrices are built: one contains the train journey times and one stores the train energy consumption at each station on a train route. The journey time matrix for one train is displayed in figure 5.19 on page 67. Rows represent departure nodes while columns represent arrival nodes.

Additionally, two graphs are drawn for supporting the timetable planner in rapidly analysing the quality of service currently offered in terms of total energy consumption on the railway line and train journey time.

The graphs obtained for the outline timetable are shown in figure 5.20 on page 68.

A1									
f_x HEADER									
	A	B	C	D	E	F	G	H	I
1	HEADER								
2	Log written: Fri Jul 15 14:40:54 BST 2016								
3	Model: \\adf.bham.ac.uk\eps\prhome24\S\SXU256\My Documents\PhD research - last develo								
4	END_HEADER								
5	A	S2	S2_1	N28	09:38:00	09:38:00	0	0	0
6	D	S2	S2_1	N28	09:38:00	09:38:00	0	0	0
7	A	S2	S2_1	N58	09:46:34	09:46:00	31.69397	9770	0
8	D	S2	S2_1	N58	09:47:34	09:47:00	31.69397	9770	0
9	A	S0	S0_1	N30	09:58:00	09:58:00	0	0	0
10	D	S0	S0_1	N30	09:58:00	09:58:00	0	0	0
11	A	S2	S2_1	N25	09:59:59	09:58:30	68.43069	26896	0
12	D	S2	S2_1	N25	10:00:59	09:59:30	68.43069	26896	0
13	A	S0	S0_1	N58	10:07:08	10:06:00	41.55729	9789.5	0
14	D	S0	S0_1	N58	10:08:08	10:07:00	41.55729	9789.5	0
15	A	S1	S1_1	N28	10:13:00	10:13:00	0	0	0
16	D	S1	S1_1	N28	10:13:00	10:13:00	0	0	0
17	A	S2	S2_1	N14	10:13:37	10:11:00	119.3173	43875	0
18	D	S2	S2_1	N14	10:14:37	10:12:00	119.3173	43875	0
19	O	S2	S2_1	N45	10:21:41	10:17:42	150.4885	52661.44	0
20	A	S1	S1_1	N58	10:22:01	10:21:00	30.11168	9770	0
21	A	S0	S0_1	N23	10:22:08	10:21:00	76.9467	26921	1
22	D	S0	S0_1	N23	10:22:08	10:21:00	76.9467	26921	1
23	D	S1	S1_1	N58	10:23:01	10:22:00	30.11168	9770	0
24	A	S2	S2_1	N50	10:30:28	10:26:30	163.8676	65201.5	0
25	D	S2	S2_1	N50	10:34:58	10:31:00	163.8676	65201.5	0
26	A	S1	S1_1	N25	10:36:05	10:33:30	64.18852	26896	0
27	D	S1	S1_1	N25	10:37:05	10:34:30	64.18852	26896	0
28	A	S3	S3_1	N24	10:38:00	10:38:00	0	0	0
29	D	S3	S3_1	N24	10:38:00	10:38:00	0	0	0

Figure 5.17: BRaVE - Log file

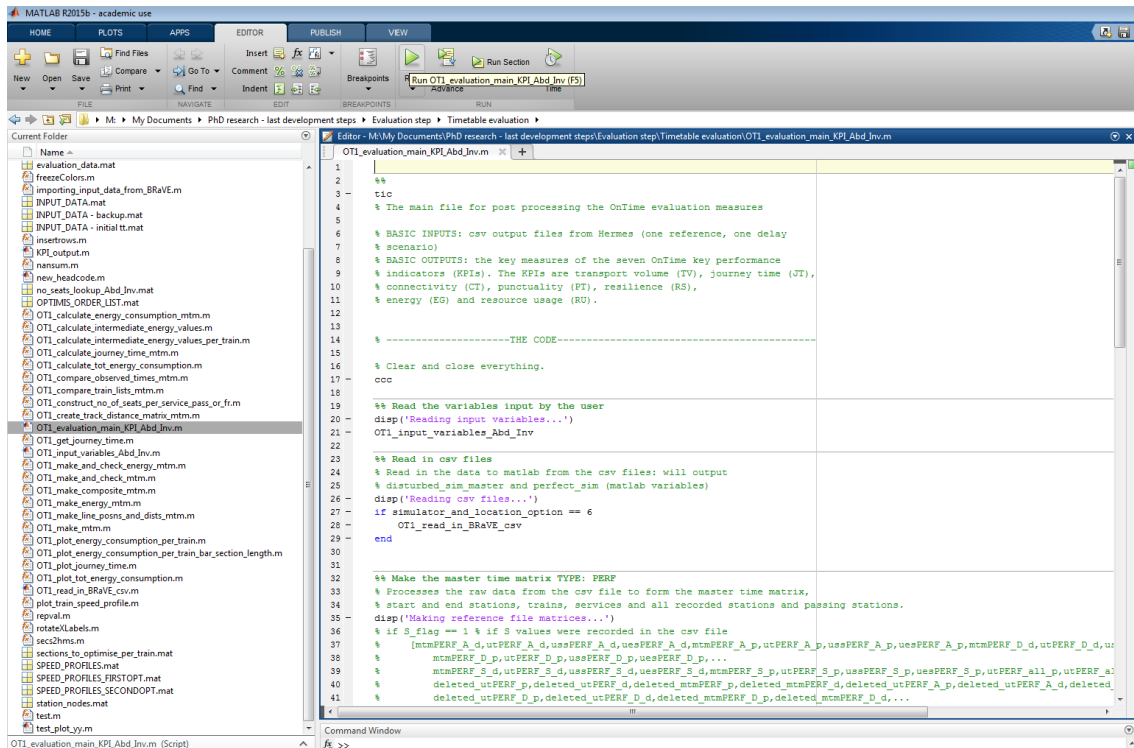


Figure 5.18: Evaluation - Log file import

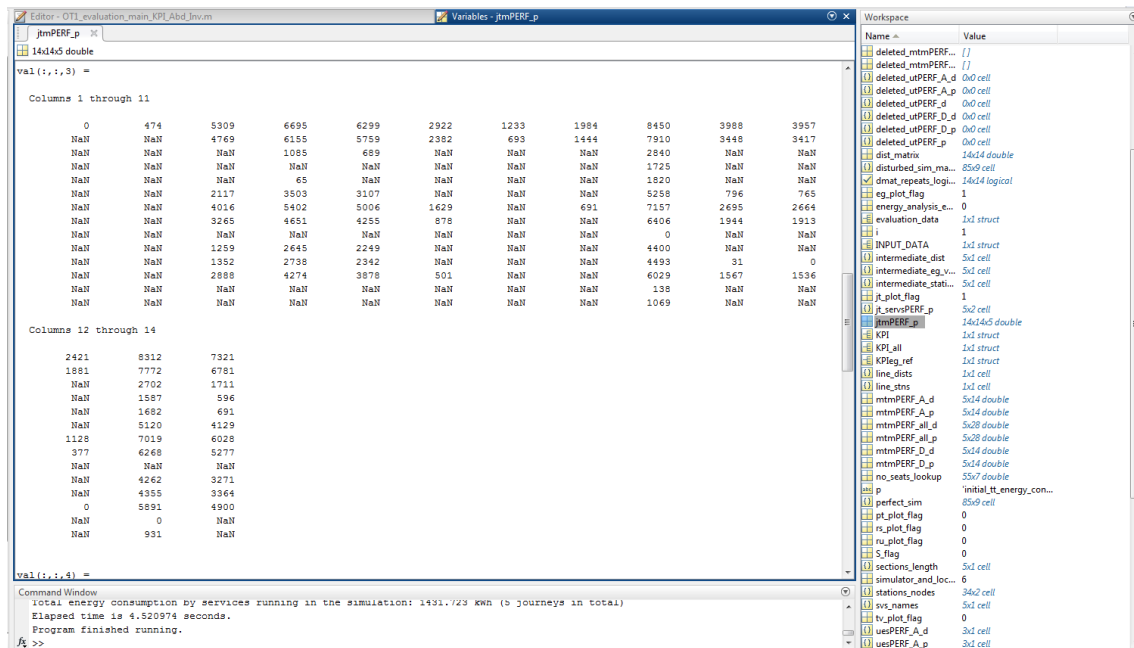
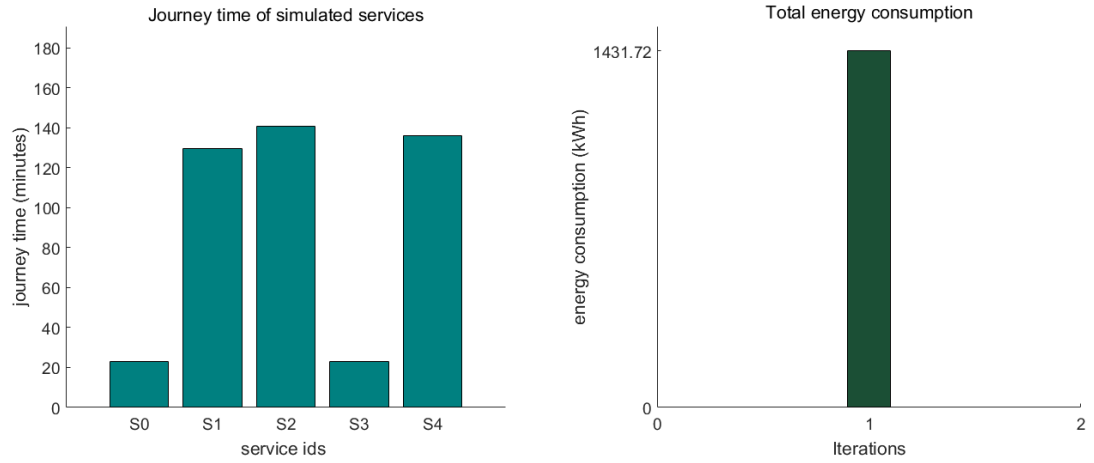


Figure 5.19: Evaluation - Partial journey time matrix



(a) Train journey time

(b) Total energy consumption consumed on the Aberdeen-Inverness line when running the set of trains

Figure 5.20: Evaluation graphs

5.1.7 Optimisation

The technical data on the railway line is transferred from the evaluation stage into the optimisation process via an XML file automatically generated by the evaluation program. An extract of the XML file produced at the first iteration of the process is given in figure 5.21 on the following page.

The cost function and the constraints of the optimisation problem have been manually set up in the optimisation program in a pre-processing phase.

The addressed optimisation problem is regarding the determination of the most appropriate train cruising speed on each inter-node section for a group of n trains, so that the total energy consumption on the railway line is minimised while trains meet operational and safety constraints. A node can be a station, a passing loop or a junction.

The problem is decomposed into single-train optimisation sub-problems in order to increase the process efficiency. Each single-train problem is solved in a pre-set sequence.


```

<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <allTrains>
  - <trainData serviceId="S2">
    <cross_locations>133150.0005018711</cross_locations>
    <cross_locations>133554.00049471855</cross_locations>
    <cross_point_list>0</cross_point_list>
    <cross_point_list>1</cross_point_list>
    <cross_stn_ids>10</cross_stn_ids>
    <cross_stn_ids>11</cross_stn_ids>
    <cross_train_names>SC195 E 50</cross_train_names>
    <cross_train_names>SC195 E 50</cross_train_names>
    <dest_location>173050.00035881996</dest_location>
    <direction>0</direction>
    <headways>360.0</headways>
    <headways>540.0</headways>
    <headways>558.0</headways>
    <headways>120.0</headways>
    <headways>450.0</headways>
    <headways>666.0</headways>
    <headways>120.0</headways>
    <headways>918.0</headways>
    <headways>564.0</headways>
    <headways>120.0</headways>
    <headways>438.0</headways>
    <headways>702.0</headways>
    <headways>120.0</headways>
    <headways_idx>1</headways_idx>
    <headways_idx>2</headways_idx>
    <headways_idx>3</headways_idx>
    <headways_idx>4</headways_idx>
    <headways_idx>5</headways_idx>
    <headways_idx>6</headways_idx>
    <headways_idx>7</headways_idx>
    <headways_idx>8</headways_idx>
    <headways_idx>9</headways_idx>
    <headways_idx>10</headways_idx>
    <headways_idx>11</headways_idx>
    <headways_idx>12</headways_idx>
    <headways_idx>13</headways_idx>
    <hw_locations>9770.0</hw_locations>
    <hw_locations>26896.000915527344</hw_locations>
    <hw_locations>43875.000915527344</hw_locations>
    <hw_locations>52275.00102996826</hw_locations>
    <hw_locations>65201.50110626221</hw_locations>
    <hw_locations>84670.00057220459</hw_locations>
    <hw_locations>85160.50057816505</hw_locations>
    <hw_locations>113971.00073075294</hw_locations>
    <hw_locations>133150.0005018711</hw_locations>
    <hw_locations>133554.00049471855</hw_locations>
    <hw_locations>148910.00018954277</hw_locations>
    <hw_locations>172215.00034213066</hw_locations>
    <hw_locations>173050.00035881996</hw_locations>
    <number_of_sections>13</number_of_sections>
    <stations_arr_time_interval>0.0</stations_arr_time_interval>
    <stations_arr_time_interval>0.0</stations_arr_time_interval>
    <stations_arr_time_interval>30.0</stations_arr_time_interval>

```

Figure 5.21: Evaluation - XML file containing the technical data on the railway network under consideration

According to the optimal train control theory (Pudney and Howlett, 1994), the optimal driving strategy on a level track is made of four regimes: acceleration at full power, cruising, coasting and braking at full power.

In this work, coasting has not been included due to software limitations. Therefore, the considered speed profile is made of three phases only, as shown in figure 5.22.

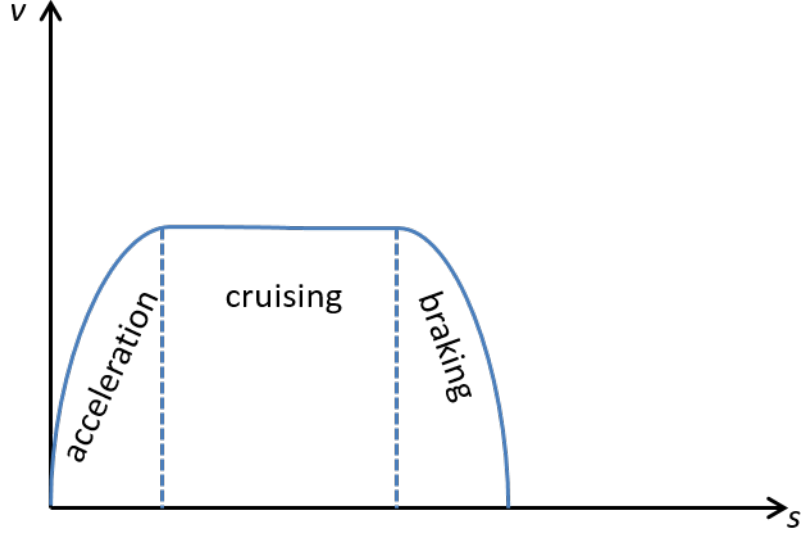


Figure 5.22: Train trajectory

A visual representation of the problem is given in figure 5.23. Each train is subject to the following constraints:

- The departure time from the origin station is fixed, for example train i leaves its first station at time $DT_{i,1}$.
- The arrival time at crossing locations, such as location k in the figure, has to occur in the time interval set by previously scheduled trains, for example train i arrival time at location k is determined by the schedule of train $(i + 1)$.
- The arrival time at the destination station is allowed to be delayed within a given time allowance according to the operational rules on the railway line; the time margin is set to 5% in this case study.

Parameter	Value	Parameter	Value
Number of variables	13	Population size	35
Crossover rate	0.8	Mutation rate	0.3
Number of elites	$\frac{Pop\ size - Crossover\ rate \times Pop\ size}{Pop\ size}$	Number of generations	10000
Avg change in the fitness function	0.000001	Time limit (s)	14400

Table 5.2: GA parameters

- The cruising speed cannot exceed the speed limit on any section of the route.
- The minimum headway time with trains ahead has to be fulfilled.
- The cruising speed is discrete and is selected from a given set.

Different speed trajectories are possible. The train cruising speed $v_{i,j}$ on each inter-node section is modified so that a train meets the defined constraints while it minimises the amount of energy consumed. The range of allowed train cruising speeds depends on the route section. The minimum train speed is used as a lower bound, whereas the section speed limit constitutes the upper bound. The speed step size is 1 km/h, although the driver may not be able to follow the trajectory at this level of accuracy. It is assumed that a human driver controls the train movement, since fully Automatic Train Operation systems are yet not available on mainline railways; a project is currently under development to install a fully automated system on the central section of the CrossRail mainline in the South East of England (MacLennan, 2012).

A microscopic simulator is used for representing the problem model and a genetic algorithm is used to solve the problem when the number of variables of the single-train optimisation problem is equal to thirteen, a brute force algorithm when only two variables form the problem. The configuration of the genetic algorithm is summarised in figure 5.2.

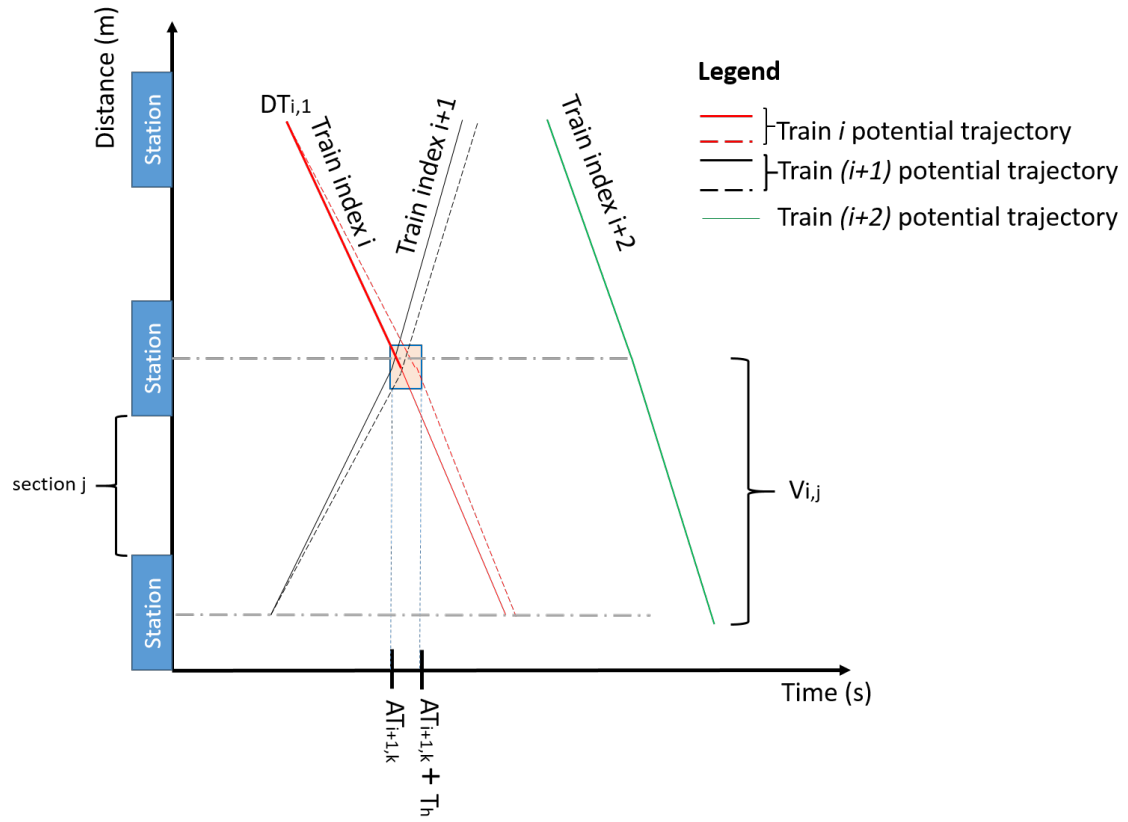


Figure 5.23: Visual representation of the optimisation problem

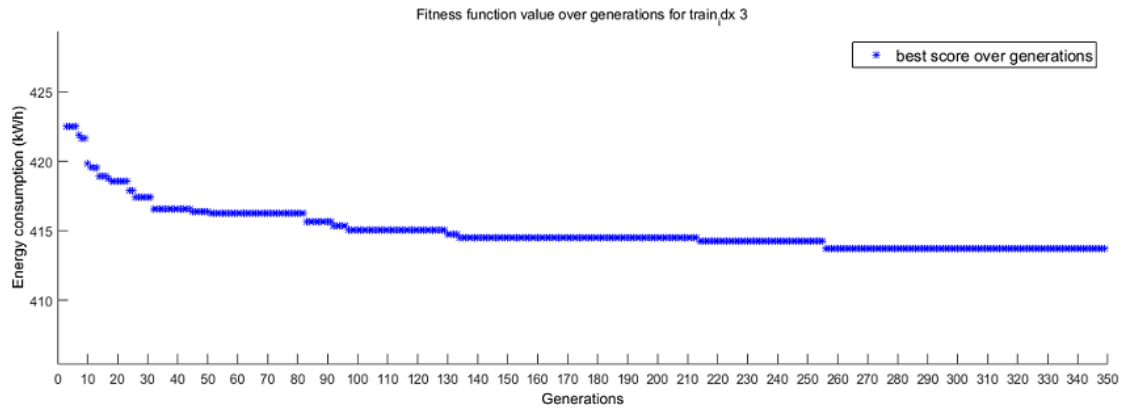


Figure 5.24: Optimisation - Fitness function convergence

The timetable planner starts the execution of the optimisation process in Matlab. At the end of the computation, the algorithm returns the best solution that it found together with a graph displaying the gradual process convergence.

The graph obtained when the cost function is regarding the minimisation of the total energy consumption on the railway line is as in figure 5.24.

5.1.8 Loop Over The Process

The optimised timetable may not be feasible. Therefore, the optimisation program automatically updates the Viriato model. The timetable planner analyses the feasibility of the timetable in Viriato and BRaVE and assess the quality of service afterwards.

If the timetable is feasible and the quality of service is satisfactory, the process terminates. In case conflicts have arisen, the optimisation program is run again for trains that have been modified in the conflict resolution process.

In the following section the results are discussed.

5.2 Numerical Results

Different optimisation configurations have been tested. These differ in the cost function that is used; the first one aims to minimise the total energy consumption of a given set of n trains:

$$\min \sum_{i=1}^n E_i, \quad (5.1)$$

while a second one includes a penalty factor to penalise deviations from the scheduled journey time:

$$\min \sum_{i=1}^n (E_i + p_i). \quad (5.2)$$

The number of trains n being optimised is equal to the number of trains being assessed.

The penalty p_i varies according to the journey time deviation percentage, as summarised in table 5.3 on the following page. The penalty calculation method is an adaptation of the work by Lu et al. (2013b). The formula used for calculating the journey time deviation is:

$$jtime_{dev} = |jtime_{act} - jtime_{sched}|/jtime_{sched},$$

where $jtime_{act}$ is the actual train journey time and $jtime_{sched}$ is the scheduled train journey time. Early arrival times are also penalised.

Journey time dev (%)	Multiplication factor (units)	Penalty p_i (kWh)
0 - 1	0	0
1 - 2.5	1	E
2.5 - 5	240	$240 \times jtime_{dev} \times E$

Table 5.3: Penalty values depending on the train journey time deviation

When the maximum allowed journey time deviation is reached, i.e. 5%, the actual energy consumption increases twelve times, so that high time deviations are discouraged. In that case, the penalty p_i would be equal to $240 \times jtime_{dev} \times E$.

When the cost function (5.1) is used, the final timetable is as in figure 5.25 on the next page. When the cost function (5.2) is employed, instead, the optimised timetable is shown in figure 5.26 on the following page.

It takes about two and a half hours to obtain a solution for one train on a desktop computer with an Intel Core i5 3.2 GHz processor and 8 GB memory.

The solutions obtained with the two cost functions are compared in terms of computational time and quality (table 5.4 on page 76).

The average computational time when cost function (5.1) is used is higher than when cost function (5.2) is employed. In terms of quality of the solution, the total energy consumption is reduced by 7% when penalties are considered and by 13% if they are not included. However, a lower energy consumption is compensated by longer journey times. As reported in table 5.4 on page 76, the algorithm tends to use all the available journey time allowance if no penalties are included in the

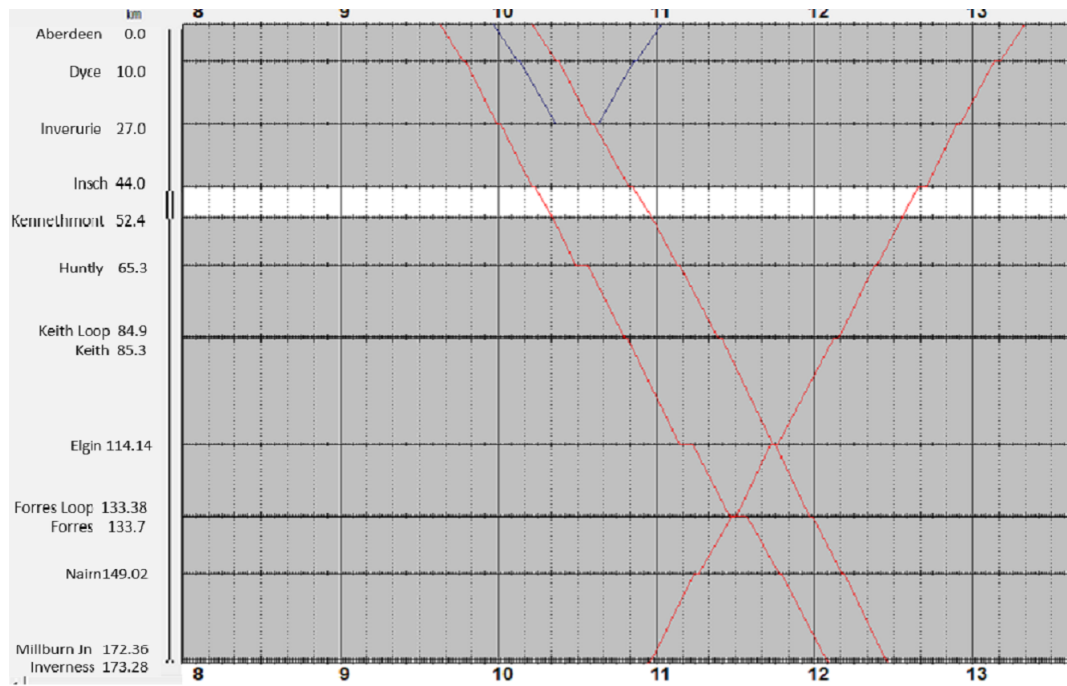


Figure 5.25: Final timetable when using cost function (5.1)

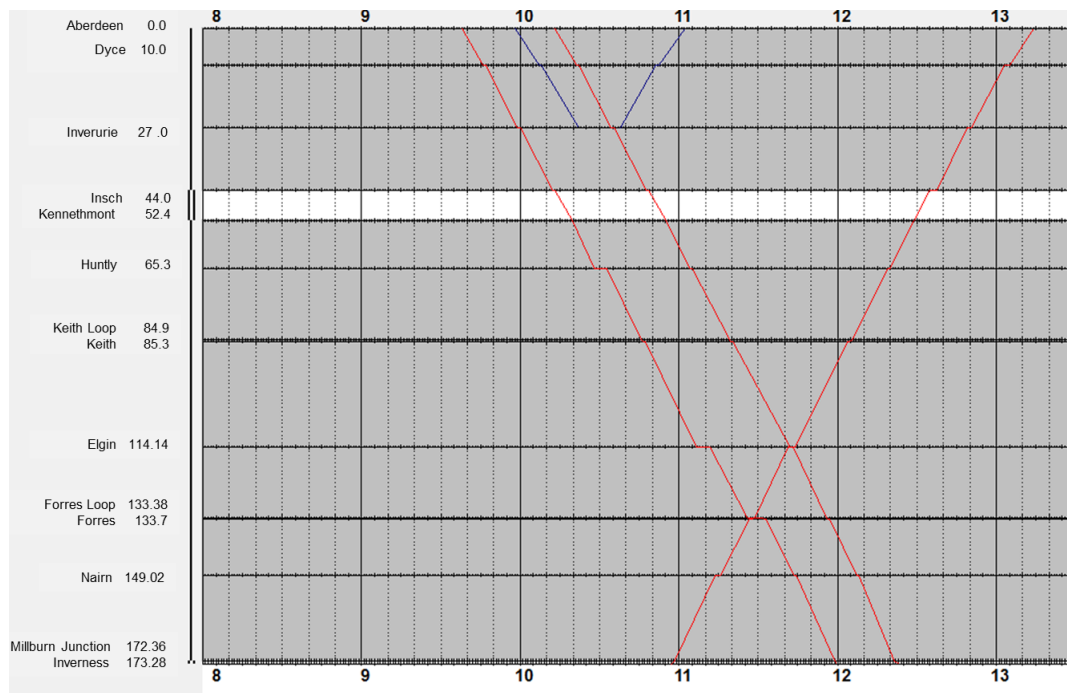


Figure 5.26: Final timetable when using cost function (5.2)

cost function. A trade-off between these two indicators has to be made in order to satisfy the train operators' requests and run trains more energy efficiently. The solution provided when using equation (5.2) on page 73, which includes penalties, is preferable. It is worth noticing that the output of the optimisation stage is a timetable and not train speed profiles. Therefore, the stated energy savings are achieved only if the proposed trajectories are followed by train drivers.

Cost function	Avg comp time (s)	Energy cons (kWh)	Energy savings (%)	Avg journey time increase (%)
eq (5.1)	9692.05	1245.51	13	5
eq (5.2)	8602.03	1331.83	7	1

Table 5.4: Optimisation results with different cost functions

5.3 Summary

In this chapter, the process flow when the framework is applied to the case study is illustrated. The results obtained are then discussed. Two optimisation cost functions are considered; one takes only energy consumption into account, while the other includes penalties for journey times that deviate up to 5% from the scheduled journey times. The results show that including penalties in the cost function leads to lower energy savings: 7% vs 13%, but to a shorter journey time. Including penalties in the cost function is believed to produce results that satisfy train operators' needs on the one hand and passenger and infrastructure managers' on the other.

In the next chapter, the conclusions that can be drawn from this research together with recommendations and suggestions for further work are given.

Chapter 6

Conclusions And Further Work

This chapter focuses on the author's achievements and choices. It then provides recommendations and suggestions for further work. These are discussed in the following sections.

6.1 Achievements

The aims of this thesis, which were stated in chapter 1 on page 1, are reported below:

- development of a method to improve timetables that makes extensive use of integrated computer systems, requiring low intervention by timetable planners.
- development of an optimisation architecture within the framework that can be used to address the optimisation needs of a given railway system.

The devised method is illustrated in figure 6.1:

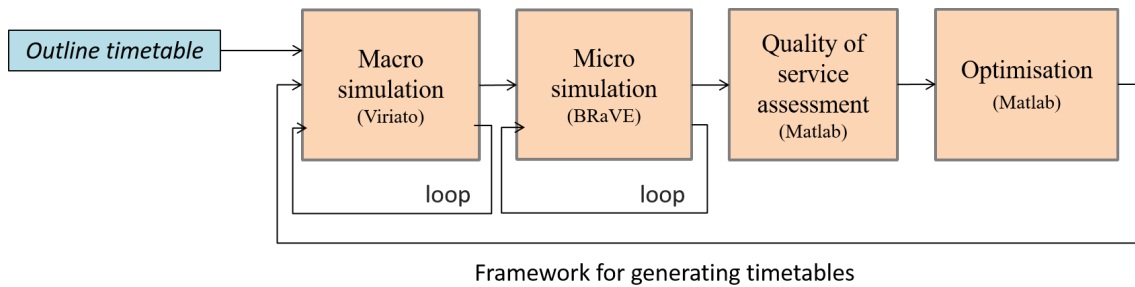


Figure 6.1: Method components

The results obtained for the case study confirm that the proposed method is successful in producing feasible and optimised timetables. In more detail, the total energy consumption on the Aberdeen-Inverness railway line for the given service is reduced by 7%, while the total journey time per train increases by 1% when compared to the initial schedule.

Moreover, the software tools used to implement the devised framework are integrated:

- BRaVE automatically reproduces a Viriato model at a microscopic level;
- the Matlab evaluation program automatically calculates the performance figures using the CSV file generated by BRaVE;
- the Matlab optimisation process automatically obtains the necessary data about the railway network from the evaluation framework via an XML file and updates the Viriato model automatically.

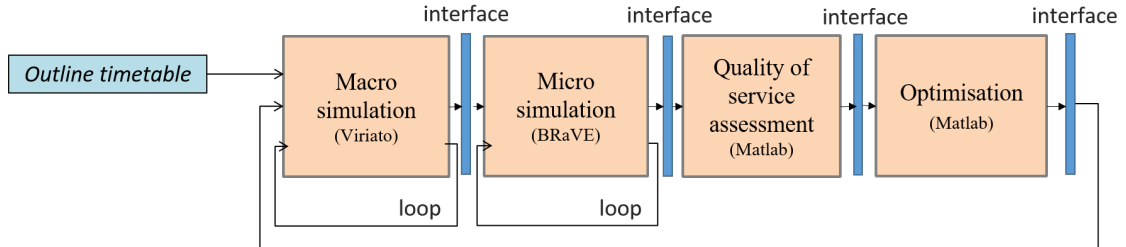


Figure 6.2: Method schematic

6.2 Recommendations

In section 2.3.4 on page 15, it is stated that using a macroscopic and a microscopic simulator, in this order, to check the feasibility of a timetable reduces the time necessary to produce a feasible timetable. This is further discussed at the end of section 3.1 on page 28, where it is explained that a macroscopic simulator enriches the microscopic feasibility function. Specifically, a macroscopic simulator detects all the macroscopic conflicts on the given network at once and determines

the impact of a resolution strategy for one conflict on the entire railway network. In addition to that, a comparison of the software components used to implement the two simulators, in section 4.4 on page 43, showed that the amount of data necessary to build a macroscopic model of a railway network is limited and does not require a detailed knowledge of the railway system being considered, whereas a microscopic simulator does. As such, setting-up a macroscopic simulator and automatically generating a microscopic model reduces both time and effort.

The following recommendation is therefore given:

- The feasibility check of timetables should be performed using an integrated macro-micro approach.

As described in section 2.2 on page 10, the framework for constructing timetables should be general, so that it can address the needs of different railway systems without requiring major modifications to its structure. This increases the scope of applicability of the framework and reduces the time necessary for setting it up. In order to meet this requirement, each step of the process should be general. Regarding the optimisation stage, as shown by the numerical results obtained for the case study, in section 5.2 on page 73, the configuration of the optimisation problem affects the performances of the algorithm and the quality of the solution. A non-problem specific optimisation framework is able to deal with different problem configurations without significant changes.

This leads to the following recommendation:

- The optimisation architecture should be general-purpose.

6.3 Further Work

In the following sections, the improvements that can be made to the proposed framework are discussed. Further development of the process implementation is discussed

together with considerations on its current efficiency and possible application to future case studies.

Software Development

The devised process still needs manual intervention by timetable planners. The tasks that are currently carried out manually are:

- Conflict resolution at a macroscopic and a microscopic level;
- configuration of the optimisation process in terms of cost function, train order, and train selection.

In order to automate the first task, both Viriato and BRaVE should be further developed. Viriato is a commercial package and, therefore, it does not allow modifications to its conflict detection module. A more flexible tool is required. According to the literature, this might lead to the necessity of developing a new tool.

Regarding the second task, instead, the indicators that form the optimisation cost function could be automatically selected by means of pre-set thresholds. This means that if an indicator figure is greater than the corresponding threshold, the indicator should be selected.

The train order could be based on a more flexible priority criterion, whereas the selection of trains that need further optimisation could be automated by checking if train arrival or departure times at relevant locations have been modified during the conflict resolution process. Finally, if the quality of service is not satisfactory, the configuration of the optimisation algorithm could be automatically changed by relaxing constraints using pre-defined rules and/or using a different train order.

Process Efficiency

The process efficiency can be enhanced; the current implementation of the optimisation algorithm requires two and a half hours on average to produce a single-train

optimised timetable. This is not computationally acceptable if a large number of trains needs to be optimised. In order to develop a more efficient process, practical knowledge could be added to the algorithm to guide it during the resolution process and/or a pre-processing stage could be included to identify the sets of feasible and infeasible solutions to the problem. Moreover, a comparison between the implemented genetic algorithm with other well-known algorithms could provide further insight into the genetic algorithm's performances.

Case Study

The Aberdeen-Inverness railway line has been chosen as a case study since it is single-track, which is interesting in terms of train interactions, and its level of complexity was appropriate for developing the devised framework. However, the service frequency on the considered railway line is low; therefore, the number of train interactions is limited. As a possible extension to this work, a busier and larger network in a perturbed scenario could be used to test the proposed method.

List of References

- Blum, C. and Roli, A. (2003). Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308.
- Bocharnikov, Y. V., Tobias, A. M., Roberts, C., Hillmanssen, S., and Goodman, C. J. (2007). Optimal driving strategy for traction energy saving on DC suburban railways. *IET Electric Power Applications*, 1(5):675–682.
- Cacchiani, V. and Toth, P. (2012). Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737.
- Caimi, G., Fuchsberger, M., Laumanns, M., and Schüpbach, K. (2011). A multi-level framework for generating train schedules in highly utilised networks. *Public Transport*, 3(1):3–24.
- Caprara, A., Galli, L., Stiller, S., and Toth, P. (2014). Delay-Robust Event Scheduling. *Operations Research*, 62(2):274–283.
- Chen, D., Ni, S., Xu, C., Lv, H., and Qin, K. (2016). A Soft Rough-Fuzzy Preference Set-Based Evaluation Method for High-Speed Train Operation Diagrams. *Mathematical Problems in Engineering*, 2016.
- Chen, L. and Roberts, C. (2012). Review of GB Railway Timetabling. Technical report, The University of Birmingham, United Kingdom.
- Chevrier, R., Marlière, G., Vulturescu, B., and Rodriguez, J. (2011). Multi-objective evolutionary algorithm for speed tuning optimization with energy saving in railway: Application and case study. *RailRome 2011*.

- Chevrier, R., Pellegrini, P., and Rodriguez, J. (2013). Energy saving in railway timetabling: A bi-objective evolutionary approach for computing alternative running times. *Transportation Research Part C: Emerging Technologies*, 37:20–41.
- De Fabris, S., Longo, G., Medeossi, G., and Pesenti, R. (2014). Automatic generation of railway timetables based on a mesoscopic infrastructure model. *Journal of Rail Transport Planning and Management*, 4(1-2):2–13.
- Dey, S., Bhattacharyya, S., and Maulik, U. (2015). Quantum behaved swarm intelligent techniques for image analysis. In *Handbook of Research on Swarm Intelligence in Engineering*, page 13. Engineering Science Reference, Hershey PA, USA.
- Douglas, J. B. (2014). SYSTRA touts RAILSIM X software package. Available at: <http://www.railwayage.com/index.php/communications/systra-touts-raimsim-x-software-package.html> (Accessed 19 April 2016).
- Elbeltagi, E., Hegazy, T., and Grierson, D. (2005). Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*, 19(1):43–53.
- Fister, I., Mernik, M., and Brest, J. (2013). Hybridization of Evolutionary Algorithms. In *Evolutionary algorithms*. Available at: <http://arxiv.org/abs/1301.0929> (Accessed: 19 April 2016).
- Ghoseiri, K., Szidarovszky, F., and Asgharpour, M. J. (2004). A multi-objective train scheduling model and solution. *Transportation Research Part B: Methodological*, 38(10):927–952.
- GOBOTiX Limited (2014). GoTRAX - Rail tracking and positioning system using visual odometry and track point detection SuperMap Challenge feasibility study. Technical report. Available at: <http://sparkrail.org> (Accessed 19 April 2016).
- Goverde, R. M. P., Besinovic, N., and Binder, A. (2015). A three - level framework for performance - based railway timetabling. *6th International Conference on Railway Modelling and Analysis*.

- Goverde, R. M. P. and Hansen, I. a. (2013). Performance indicators for railway timetables. *IEEE ICIRT 2013 - Proceedings: IEEE International Conference on Intelligent Rail Transportation*, pages 301–306.
- Goverde, R. M. P. and Odijk, M. (2002). Performance evaluation of network timetables using PETER. *Computers in Railways VIII*, pages 731–740.
- Health Market Science, I. (2013). Jackcess. Available at: <http://jackcess.sourceforge.net> (Accessed 19 April 2016).
- Holland, J. H. (1992). Genetic Algorithms - Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand.
- Jiang, Z., Hsu, C. H., Zhang, D., and Zou, X. (2016). Evaluating rail transit timetable using big passengers' data. *Journal of Computer and System Sciences*, 82(1):144–155.
- Kettner, M., Sewcyk, B., and Eickmann, C. (2003). Integrating Microscopic and Macroscopic Models for Railway Network Evaluation. In *Proceedings of European Transport Conference*.
- Kroon, L., Huisman, D., Abbink, E., Fioole, P.-J., Fischetti, M., Maroti, G., Schrijver, A., Steenbeek, A., and Ybema, R. (2009). The New Dutch Timetable: The OR Revolution. *Interfaces*, 39(1):6–17.
- Kunimatsu, T., Hirai, C., and Tomii, N. (2012). Train timetable evaluation from the viewpoint of passengers by microsimulation of train operation and passenger flow. *Electrical Engineering in Japan*, 181(4):51–62.
- Li, X. and Lo, H. K. (2014a). An energy-efficient scheduling and speed control approach for metro rail operations. *Transportation Research Part B: Methodological*, 64:73–89.
- Li, X. and Lo, H. K. (2014b). Energy minimization in dynamic train scheduling and control for metro rail operations. *Transportation Research Part B: Methodological*, 70:269–284.

- Li, X., Wang, D., Li, K., and Gao, Z. (2013). A green train scheduling model and fuzzy multi-objective optimization algorithm. *Applied Mathematical Modelling*, 37(4):2063–2073.
- Lu, M., Nicholson, G. L., Schmid, F., Dai, L., Chen, L., and Roberts, C. (2013a). A Framework for the Evaluation of the Performance of Railway Networks. *International Journal of Railway Technology*, 2(2):79–96.
- Lu, S., Hillmansen, S., Ho, T. K., and Roberts, C. (2013b). Single-Train Trajectory Optimization. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):743–750.
- Lucchini, L., Curchod, A., and Rivier, R. (2001). Transalpine rail network: A capacity assessment model (CAPRES). In *First Swiss Transport Research Conference*.
- MacLennan, P. (2012). Crossrail awards signalling system contract. Available at: <http://www.crossrail.co.uk/news/articles/crossrail-awards-signalling-system-contract> (Accessed 20 April 2016).
- Marinov, M., Sahin, I., Ricci, S., and Vasic-Franklin, G. (2013). Railway operations, timetabling and control. *Research in Transportation Economics*, 41(1):59–75.
- Marinov, M. and Viegas, J. (2011). A mesoscopic simulation modelling methodology for analyzing and evaluating freight train operations in a rail network. *Simulation Modelling Practice and Theory*, 19(1):516–539.
- Maroti, G. (2006). *Operations research models for railway rolling stock planning*. PhD thesis, TU Eindhoven.
- Michalewicz, Z. (1996). GAs: What are they? In *Genetic algorithms + Data structures = Evolution programs*, pages 13–31. Springer, Berlin; Heidelberg; New York, 3rd edition.
- Middlekoop, D. and Bouwman, M. (2001). SIMONE: large scale train network simulations. In *Proceedings of the 2001 Winter Simulation Conference*.

- Müller-Hannemann, M. and Shirra, S. (2010). *Algorithm engineering: bridging the gap between algorithm theory and practice*. Springer, Berlin; Heidelberg.
- Nash, A. and Huerlimann, D. (2004). Railroad simulation using OpenTrack. *Computers in Railways IX*, pages 45–54.
- Network Rail (2014). Scotland sectional appendix. Available at: <http://www.networkrail.co.uk/browse%20documents/sectional%20appendix/scotland%20sectional%20appendix.pdf> (Accessed 20 April 2016).
- Network Rail (2016). Timetable planning rules. Available at: <http://www.networkrail.co.uk/aspx/3741.aspx> (Accessed 20 April 2016).
- Nicholson, G. L., Kirkwood, D., Roberts, C., and Schmid, F. (2015). Benchmarking and evaluation of railway operations performance. *Journal of Rail Transport Planning and Management*, 5(4):274–293.
- Oracle (2003). JABX. Available at: <http://www.oracle.com/technetwork/articles/javase/index-140168.html> (Accessed 20 April 2016).
- Panou, C., Stergidou, A., Emery, D., Tzieropoulos, P., Goverde, R., Toth, P., Cacchiari, V., Albrecht, T., and Binder, A. (2013). Assessment of State-of-Art of Train Timetabling. Technical report. Available at: <http://www.ontime-project.eu/documents.aspx> (Accessed 20 April 2016).
- Pouryousef, H., Lautala, P., and White, T. (2015). Railroad capacity tools and methodologies in the U.S. and Europe. *Journal of Modern Transportation*, 23(1):30–42.
- Pudney, P. and Howlett, P. (1994). Optimal Driving Strategies for a Train Journey With Speed Limits. *Journal of the Australian Mathematical Society Series B-Applied Mathematics*, 36(1):38–49.
- Putallaz, Y. and Rivier, R. (2004). Strategic evolution of railways corridors infra-structure: dual approach for assessing capacity investments and M&R strategies. In *Computers in Railways IX*, pages 61–72.

- Radtke, A. and Hauptmann, D. (2004). Automated planning of timetables in large railway networks using a microscopic data basis and railway simulation techniques. In *Computers in Railways IX*, pages 615–625.
- Reliable Data Systems (2014). FuTRO supermap challenge : Driver support system feasibility study. Technical report. Available at: <http://sparkrail.org> (Accessed 19 April 2016).
- RMCON (2016). RailSys 10. Available at: <http://www.rmcon.de> (Accessed 20 April 2016).
- RSSB (2013). FuTRO - GB Timetable Optimisation - baselining the challenge. Technical report. Available at: <http://sparkrail.org> (Accessed 20 April 2016).
- Scheepmaker, G. M. and Goverde, R. M. P. (2015). The interplay between energy-efficient train control and scheduled running time supplements. *Journal of Rail Transport Planning & Management*, 5(4):225–239.
- Schittenhelm, B. and Landex, A. (2010). Computation of a suburban night train timetable based on key performance indicators. In *Computers in Railways XII*, pages 923–934.
- Schlechte, T. (2011). Railway Track Allocation - Simulation and Optimization. In *Proceedings of the 4th International Seminar on Railway Operations Modelling and Analysis*.
- Sels, P., Cattrysse, D., and Vansteenwegen, P. (2015). Practical Macroscopic Evaluation and Comparison of Railway Timetables. *Transportation Research Procedia*, 10:625–633.
- Siefer, T. (2008). Simulation. In *Railway Timetable & Traffic*, pages 155–169. Eurail Press, Hamburg, Germany.
- SIEMENS (2007). Falko Design and validation of timetables. Available at: <http://www.mobility.siemens.com/mobility/global/sitecollectiondocuments/en/rail-solutions/rail-automation/operations-control-systems/falko-en.pdf> (Accessed 20 April 2016).

- SMA and Partner (2009). Viriato users manual. Technical report, Zurich, Switzerland.
- Su, S., Li, X., Tang, T., and Gao, Z. (2013). A Subway Train Timetable Optimization Approach Based on Energy-Efficient Operation Strategy. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):883–893.
- Takagi, R. (2012). Newly developed simple railway timetable evaluation program Sujic with the new model to deal with re-scheduling. In *Computers in Railways XIII*, pages 513–520.
- Trapeze Group (2016). Trapeze Rail System Software for modelling and Planning. Technical report. Available at: <http://www.trapezegroup.co.uk/> (Accessed 20 April 2016).
- TSLG (2012). Rail technical strategy. Available at: <http://www.rssb.co.uk/future-railway-programme/railway-of-the-future> (Accessed 20 April 2016).
- Tutcher, J., Easton, J., Roberts, C., Myall, R., Hargreaves, M., and Tiller, C. (2013). Ontology-based data management for the GB rail industry: Feasibility study. Technical report. Available at: <http://www.sparkrail.org> (Accessed 19 April 2016).
- VIA Consulting & Development GmbH (2016). OnTime - network wide analysis of timetable stability. Available at: www.ontime-rail.com (Accessed 20 April 2016).
- W3C (1994). XML language. Available at: <http://www.w3.org/> (Accessed 20 April 2016).
- Wang, Y. (2014). *Optimal Trajectory Planning and Train Scheduling for Railway Systems*. PhD thesis, Technical University Delft.
- Watanabe, S. and Koseki, T. (2015). Energy-saving train scheduling diagram for automatically operated electric railway. *Journal of Rail Transport Planning & Management*, 5(3):183–193.
- Watson, R. (2001). The effect of railway privatization on train planning: a case study of the UK. *Transport Reviews*, 21(2):181–193.

- Xu, X., Li, K., and Li, X. (2016). A multi-objective subway timetable optimization approach with minimum passenger time and energy consumption. *Journal of Advanced Transportation*, 50(1):69–95.
- Yang, L., Li, K., Gao, Z., and Li, X. (2012). Optimizing trains movement on a railway network. *Omega*, 40:619–633.
- Yang, X., Li, X., Ning, B., and Tang, T. (2015). An optimisation method for train scheduling with minimum energy consumption and travel time in metro rail systems. *Transportmetrica B: Transport Dynamics*, 3(2):79–98.

Appendices

Appendix A

Optimisation models and algorithms

In the following paragraphs, possible models and algorithms for the optimisation problem addressed in the case study are discussed.

Problem Model

Three possible models are described and compared.

Mathematical Model

The optimisation problem can be classified as an integer programming problem. A corresponding model is developed. The variables and the parameters of the model are described in table A.1 on the next page.

Variable	Description
I	Train index $ I = n$
J	Section index $ J = m$
K	Location index $ K = p$
H	Subsection index $ H = ns_j$
$dt_{i,1}$	Departure time of train i from the first station it serves on a given journey
$at_{i,k}$	Arrival time of train i at location k
$v_{i,j,h}$	Cruising speed of train i on subsection h of section j
$eg_{i,j,h}(v_{i,j,h})$	Energy consumed by train i to travel subsection h of section j

Table A.1: Model variables

Parameter	Description
$DT_{i,1}$	Departure time of train i from the first station it serves on a given journey
$AT_{i,k}$	Scheduled arrival time of train i at location k
T_h	Arrival time tolerance at location k
$HT_{(i,i+2),j}$	Headway time between following trains i and $(i+2)$
$V_{i,j,h}^{max}$	Line speed on subsection h of section j
$SV_{i,j,h}$	Set of allowed integer speeds for train i on subsection h of section j

Table A.2: Model parameters

The developed model is shown below:

$$\min z = \sum_{i=1}^n \sum_{j=1}^{m_i} \sum_{h=1}^{ns_j} eg_{i,j,h}(v_{i,j,h}), \quad (\text{A.1})$$

$$dt_{i,1} = DT_{i,1} \quad \forall i \mid i = 1, \dots, n \quad (\text{A.2})$$

$$AT_{q,k} \leq at_{i,k} \leq AT_{q,k} + t_h \quad \forall i, k, q \mid i = 1, \dots, n; \quad q = 1, \dots, n \mid i \neq q; \quad k = 1, \dots, p \quad (\text{A.3})$$

$$at_{i,j} - at_{i+2,j} > HT_{(i,i+2),j} \quad \forall i, j \mid i = 1, \dots, n \quad j = 1, \dots, m_i \quad (\text{A.4})$$

$$v_{i,j,h} \leq V_{i,j,h}^{max} \quad \forall j, h \mid j = 1, \dots, m_i \quad h = 1, \dots, ns_j \quad (\text{A.5})$$

$$v_{i,j,h} \in SV_{i,j,h} \quad (\text{A.6})$$

A given railway line is split into m inter-node sections and involves n trains, each of which runs over $m_i \leq m$ sections. The line speed limit can vary over a section; therefore, a section is split into subsections, one per speed limit, in order to accurately estimate the train energy consumption and journey time. It is assumed that the same speed limits are imposed in both directions. The cost function is about minimising the total energy consumption of the n trains on their journey from origin to destination stations eq. (A.1). According to constraint (A.2), a train departure time from its origin station is fixed. Constraints (A.3) and (A.4) impose the satisfaction of a train arrival time at important timing points within a time threshold (A.3) and the headway time between following trains travelling in the same direction (A.4). Finally, constraints (A.5-A.6) represent the fact that the cruising speed of

a train on each subsection should be not greater than the line speed limit on that subsection (A.5) and that speeds are natural numbers chosen from a defined set of speeds (A.6).

The calculation of the energy consumption of a train on a subsection neglects acceleration and braking phases. Moreover, it is assumed that the line is without gradients, in order to reduce the complexity of the formulation. As a consequence, the energy consumed by a train on a subsection is due to the resistance force that has to be balanced to move it at cruising speed over the subsection length $l_{j,h}$:

$$eg_{i,j,h}(v_{i,j,h}) = F_{i,j,h} \times l_{j,h}$$

where $F_{i,j,h}$ is made of the train resistance forces, which are expressed using the Davis equation with parameters R_0^i, R_1^i, R_2^i :

$$F_{i,j,h} = R_0^i + R_1^i \times v(i, j, h) + R_2^i \times v(i, j, h)^2$$

The journey time required by a train to travel across a subsection at cruising speed is determined using the formula for linear motion over the subsection length $l_{j,h}$:

$$jt_{i,j,h} = \frac{l_{j,h}}{v_{i,j,h}}$$

The arrival time of a train at the end of section j^* , which can be a timing point k , can then be calculated from the departure time of the train at its origin station as the sum of the journey times on each subsection up to the considered timing point:

$$at_{i,j^*} = at_{i,k} = DT_i + \sum_{j=1}^{j^*} \sum_{h=1}^{ns_j} jt_{i,j,h}.$$

Although a mathematical model allows a concise definition of the problem, its complexity increases as soon as a more realistic representation of the problem is sought. Moreover, neglecting the acceleration and braking phases in the calculation results in overlooking the train energy consumption.

Micro-Simulator

A micro-simulator reproduces train movements on a railway network using a fine-grained model of the infrastructure. As time progresses, the position of a train is calculated together with its energy consumption and running time. Given the accuracy used to represent the railway system, the estimation of train energy and running time is reliable. However, if a high number of trains are considered, the simulation can be slow. In (Yang et al., 2012), the authors address the minimisation of energy consumption and journey time deviations from target values on a railway network; simulation procedures have been developed in the solving process in order to overcome the difficulties of analytical calculations. However, it is stated that the simulation process is time consuming, particularly if a high number of trains and tracks are involved. This model meets the requirement of providing accurate results; however, the computational time increases with the accuracy of the simulation. In order to use it, the computational time has to be reasonable. Possible ways to reduce the time complexity are the use of a variable time step or heuristic rules based on conventional operational best practice.

Graph-Based Model

A graph G is an ordered pair $G = (V, E)$ of a set V of vertices and a set E of edges. An edge between two vertices indicates a relation between these vertices and its meaning depends on the considered problem. Additional information can be added to vertices or edges in the form of weights. Graph-based models are easy to understand and, due to their structure, they enable the development of efficient algorithms. However, the main disadvantage of the model is its difficulty in representing the complex constraints that can characterise a problem (Müller-Hannemann and Shirra, 2010). Regarding the problem addressed in this thesis, a graph-based model is not capable of representing the energy that a train consumes during the transition between consecutive sections, thus giving an imprecise estimation of the total train energy consumption on its route.

Model Choice

In table A.3 on the following page, the main advantages and disadvantages of the different models are summarised. A microscopic simulator was chosen in this work, because it met the required level of accuracy.

Model	Advantages	Disadvantages	Source
Integer programming model	Concise definition of the problem	The complexity of the model increases as soon as more realistic aspects of the system are included	See (Yang et al., 2015) for an example
Micro-simulation model	Realistic representation of train movements	The computational time increases with accuracy	(Yang et al., 2012)
Graph-based model	Easy to understand	It enables the development of computationally efficient algorithms, but problem constraints cannot always be represented in a graph	(Müller-Hannemann and Shirra, 2010)

Table A.3: Model comparison

Algorithms

Trains can be optimised simultaneously or sequentially; this means that they are optimised at the same time or in order of priority. Wang (2014), in her thesis, compares the quality of the solution and the computational time of the two approaches when minimising the total energy consumption of two following trains subject to constraints imposed by the train characteristics and the signalling system. The size of the problem in terms of number of variables and, thus, the computational time, when a simultaneous approach is used, is greater than the one when a sequential procedure is used. However, the quality of the solution is in general better since trains are considered at the same time. The authors suggest that a distributed approach might be more efficient when a large number of trains has to be optimised.

The following sections describe potential algorithms to address the problem of this thesis.

Non-Problem Specific Optimisation Algorithms

The objective is to find a good quality solution, but not necessarily the optimum, in a reasonable computational time. The algorithm should be efficient, scalable and non-problem specific.

A common classification distinguishes between complete and approximate algorithms. Complete algorithms enumerate all the solutions until they find the optimum one, whereas approximate ones may return a local optimum, but in a practical computational time (Blum and Roli, 2003).

Given the desired characteristics of the algorithm to be used, only approximate algorithms are analysed in the following. These algorithms can be classified as single-point or population-based search methods; the first ones work on a single solution, whereas the second ones evolve multiple solutions simultaneously. Population-based search methods are more efficient than single-point ones (Dey et al., 2015). Therefore, only these methods are discussed in the following.

Evolutionary algorithms are a macro-category of population-based search methods. These are general, thus they can be applied to many optimisation problems, and often provide adequate solutions (Fister et al., 2013). Moreover, they usually deal with large-scale optimisation problems (Elbeltagi et al., 2005). Therefore, they meet the requirements of scalability and generality. Well-known evolutionary algorithms are: Genetic Algorithms (GAs), Particle Swarm Optimisation (PSOs) and Ant Colony Optimisation algorithms (ACOs).

In (Elbeltagi et al., 2005), a concise description of each algorithm is provided together with a comparison in terms of quality of solution and computational time when applied to a discrete optimisation problem. The results show that, on average, the quality of the solution of PSOs is the highest, while the one of ACOs is the lowest,

whereas the computational time of ACOs is the lowest and the one of GAs is the highest. They conclude that PSOs show the overall best performances.

Problem-Specific Optimisation Algorithms

In this section, the algorithms that have been proposed in order to address a similar problem to the one of the case study of this thesis are described below.

In (Lu et al., 2013b), the addressed problem is about minimising the energy consumption of a single train while it meets the punctuality constraint. To solve that, the train speed at specific locations along the route is selected from a list of possible speeds. The authors compare three algorithms: Dynamic Programming (DP), Genetic Algorithm (GA) and Ant Colony Optimisation algorithm (ACO). The results show that ACO uses the minimum computational time, whereas DP finds the solution of best quality. The authors found that the GA can perform poorly when a large journey time is allowed.

In (Bocharnikov et al., 2007), the authors propose a method for calculating traction and regenerative braking forces and a coasting factor to efficiently use energy and satisfy timetable constraints. Cruising speed is not considered in this work. A genetic algorithm is chosen in their work with the following advantages being cited: avoidance of local minima, high probability of finding a near-optimal solution to large problems in a relatively small number of generations, insensitivity to non-linearities, easiness of implementation and flexibility. However, no figures are given regarding the algorithm's computational time.

In (Su et al., 2013) a method is described, which is able to calculate the optimum speed profile on a section between two consecutive stations on a subway line. The proposed method analytically determines, on each section, the optimal cruising and braking speeds and the cruising, coasting and braking points. The method has proved to be efficient on numerical tests, although variable gradients, variable traction and braking forces have to be further studied. Moreover, a subway line

forms a distinct case, since metro vehicles generally have similar performances and the stopping pattern is fixed.

In (Chevrier et al., 2011) an evolutionary algorithm is implemented in order to tune the train speed on each inter-station section so that its journey time, delay and energy consumption are minimised. The method has proved to be efficient.

Finally, in (Scheepmaker and Goverde, 2015), a bisection method and a Fibonacci search algorithm are used together to determine the optimal coasting point and the optimal cruising speed to minimise the total train traction energy consumption with fixed total journey time. The authors state that the algorithm's efficiency needs improvement.

Possible Choice

As concluded at the end of the section on generic optimisation algorithms, a PSO algorithm is a promising choice. The literature on the energy optimisation problem in the railway domain has not tried to implement a PSO algorithm. They suggest that evolutionary algorithms are suitable for this type of problem since they are efficient and general. A comparison between ACOs and GAs by Lu et al. (2013b) reveals that an ACO algorithm provides a worse solution than a GA, but its computational time is shorter. A trade-off has to be sought.

The choice depends on the desired trade-off. In this work, the solution quality was considered as very important, therefore a GA was chosen.

Appendix B

Pseudo code of the main parts of the framework

In the following pages the pseudo-code that has been developed to interface the software packages is shown.

```

/* BRaVE interface that imports a Viriato model */
public class ViriatoImporter {

/* the following method includes the functions that build the components of a
BRaVE model */

    public ViriatoImporter{}

    private void loadModel(file) {

        readDatabase(file);

        createStations();
        createBranches();
        connectBranchesToNodes();
        createSections();

        wirePoints();
        createSignals(TWO_ASPECT_SIG);

        createBlockSectionRoutes(empty parameters);

        createTrains();
        createServices();

        generateInterlocking();
    }

/* Reading the Viriato tables from the given Viriato database file*/
    private void readDatabase(file) {

        // opening the database
        viriatoDatabase = Database.open(file);

        // reading the data on node position, separation times, number of platforms,
        // access restrictions due to the node layout
        readNodesDefinitionTable();
        readNodesPlatformsDefinitionTable();
        readNodeRestrictRelationTable();

        // reading the data on section length, gradient profile and speed limits
        readSectionDefinitionTable();
        readSectionProfileDefinitionTable(pmStart, pmStart + pmStep);
        readSectionSpeedDefinitionTable(pmStart, pmStart + pmStep);

        // reading the data on train maximum speed, weight, length, schedule and
        // route
        readRollingStockDefinitionTable(pmStart, pmStart + pmStep);
        readTrainDefinitionTable(pmStart, pmStart + pmStep);
    }
}

```

```

/* Reading the Viriato table relative to the definition of nodes */
private void readNodesDefinitionTable(){

    Table nodesDefinitionTable = viriatoDatabase.getTable("Nodes Definition");

    for(row: nodesDefinitionTable) {

        // node Id
        String nodeId = (String) row.get("NodeID");

        // node position
        xCoordinate = (Float)row.get("X_Coordinate");
        yCoordinate = (Float)row.get("Y_Coordinate");

        // separation times inside a node so that points can be set and the route has been
        // cleared
        sepTimeArrArr = (Float)row.get("SepTimeArrArr");
        sepTimeArrDep = (Float)row.get("SepTimeArrDep");
        sepTimeArrDepPass = (Float)row.get("SepTimeArrDepPass");
        sepTimeDepArr = (Float)row.get("SepTimeDepArr");

        // storing the node in a BRaVE data structure
        nodeDefinitions.put(nodeId, new ViriatoNodeDefinition(
            nodeId, nodeName, xCoordinate, yCoordinate,
            sepTimeArrArr, sepTimeArrDep, sepTimeArrDepPass,
            sepTimeDepArr));
    }
}

/* Creating the tracks inside each station (or passing loop) */
private void createStations() {

    nodeIds = nodeDefinitions.list();

    while(nodeIds.hasNext()) {
        nodeId = nodeIds.next();

        viriatoNode = nodeDefinitions.get(nodeId);

        // Node position
        x = viriatoNode.getXCoordinate();
        y = viriatoNode.getYCoordinate();

        x -= STATION_WIDTH / 2;
        y -= LINE_SPACING * 3;

        // Creating the station tracks for each node platform
        for(nodePlatform : viriatoNode.getNodePlatformsDefinitions()) {

            node1 = nodeManager.create("N" + (++nodeCount));
            node1.setNetPoint(x, y);
            node1.setStationID(viriatoNode.getNodeID());
        }
    }
}

```

```

        node2 = nodeManager.create("N" + (++nodeCount));
        node2.setNetPoint(x + STATION_WIDTH, y);
        node2.setStationID(viriatoNode.getNodeID());

        // station track
        path1 = pathManager.create();
        path1.setStartNode(node1.getName());
        path1.setEndNode(node2.getName());
        path1.setLength(nodePlatform.getLength());

        y -= LINE_SPACING;
    }
}

}

/* Creating the branches on each side of a station (or passing loop) */
private void createBranches() {

    nodeIDs = nodeDefinitions.list();

    while(nodeIDs.hasNext()) {

        nodeId = nodeIDs.next();

        viriatoNode = nodeDefinitions.get(nodeId);

        //defining the position of the branch path
        leftCoordinateX = viriatoNode.getxCoordinate() - (STATION_WIDTH / 2) - BRANCH_LENGTH;
        leftCoordinateY = viriatoNode.getyCoordinate() - (LINE_SPACING*3);

        rightCoordinateX = viriatoNode.getxCoordinate() + (STATION_WIDTH / 2) + BRANCH_LENGTH;
        rightCoordinateY = viriatoNode.getyCoordinate() - (LINE_SPACING*3);

        // defining which branches should be to the left of a station track and which
        // ones should be to the right of the station track
        for(branch : viriatoNode.getNodeBranchDefinitionsList()) {
            if(branch.getDrawDirection() == -1) {
                branchesOnLeft.add(branch);
            } else {
                branchesOnRight.add(branch);
            }
        }

        startX = leftCoordinateX - BRANCH_LENGTH;
        endX = leftCoordinateX;
        y = leftCoordinateY;

        for(branch : branchesOnLeft) {

            node1 = nodeManager.create("N" + (++nodeCount));
            node1.setNetPoint(startX, y);

```

```

        node2 = nodeManager.create("N" + (++nodeCount));
        node2.setNetPoint(endX, y);

        path1 = pathManager.create();
        path1.setStartNode(node1.getName());
        path1.setEndNode(node2.getName());

        y -= LINE_SPACING;
    }

    startX = rightCoordinateX + BRANCH_LENGTH;
    endX = rightCoordinateX;
    y = rightY;

    for(branch : branchesOnRight) {

        node1 = nodeManager.create("N" + (++nodeCount));
        node1.setNetPoint(startX, y);
        node2 = nodeManager.create("N" + (++nodeCount));
        node2.setNetPoint(endX, y);

        path1 = pathManager.create();
        path1.setStartNode(node1.getName());
        path1.setEndNode(node2.getName());

        y -= LINE_SPACING;
    }
}

/* Creating connections between branches and nodes*/
private void connectBranchesToNodes() {

    nodeIDs = nodeDefinitions.list();

    while(nodeIDs.hasNext()) {

        nodeId = nodeIDs.next();

        viriatoNode = nodeDefinitions.get(nodeId);

        for(branch : viriatoNode.getNodeBranchDefinitionsList()) {

            branchId = branch.getBranchName();
            leftSide = branch.getDrawDirection() == -1;

            // for each track inside a node a left and a right connection are drawn
            for(nodePlatform : viriatoNode.getNodePlatformsDefinitions()) {

```

```

platformId = nodePlatform.getPlatformID();

allowedExit = !viriatoNode.hasRestrictionFor(platformId, branchId);
allowedEntry = !viriatoNode.hasRestrictionFor(platformId,
branchId);

if(allowedExit || allowedEntry) {

    connectionLength = getConnectionLength();

    // drawing the connector nodes
    connectorNode1 = nodeManager.create("N" + (+
+nodeCount));
    connectorNode1X = leftSide ?
branch.getLocalEndNode().getNetPoint().x +
MICROSCOPIC_POINTS_SIZE :
branch.getLocalEndNode().getNetPoint().x -
MICROSCOPIC_POINTS_SIZE;
    connectorNode1.setNetPoint(connectorNode1X,
branch.getLocalEndNode().getNetPoint().y);

    connectorNode2 = nodeManager.create("N" + (+
+nodeCount));
    connectorNode2X = leftSide ?
nodePlatform.getLocalLeftNode().getNetPoint().x -
MICROSCOPIC_POINTS_SIZE :
nodePlatform.getLocalRightNode().getNetPoint().x +
MICROSCOPIC_POINTS_SIZE;
    connectorNode2.setNetPoint(connectorNode2X,
nodePlatform.getLocalLeftNode().getNetPoint().y);

    // drawing the path from the branch to connectorNode1
    path1 = pathManager.create();
    path1.setStartNode(branch.getLocalEndNode());
    path1.setEndNode(connectorNode1);
    path1.setLength(0);

    // drawing the path from connectorNode1 to
connectorNode2 - the actual connection link
    path2 = pathManager.create();
    path2.setStartNode(connectorNode1);
    path2.setEndNode(connectorNode2);
    path2.setLength(connectionLength);

    // drawing the path from connectorNode2 to the track inside
a node
    path3 = pathManager.create();
    path3.setStartNode(connectorNode2);
    path3.setEndNode(leftSide ?
nodePlatform.getLocalLeftNode():
nodePlatform.getLocalRightNode());

}
}
}
}

```



```

}

/* Creating the sections between main nodes */
private void createSections(double pmStart, double pmStop) {

    for(sectionMain : sectionDefinitions.list()) {

        fromNode = sectionMain.getStartNode();

        while(fromNode != null) {

            sectionDefinition = sectionMain.getSectionFromNode(fromNode);

            toNode = sectionDefinition.getNextNode();

            numberOfLines = sectionDefinition.getNumberOfTracks();

            sourceNode = nodeDefinitions.get(fromNode);
            targetNode = nodeDefinitions.get(toNode);

            yPos = sourceNode.getyCoordinate();

            // connector nodes at the source are drawn
            for(j=0; j<numberOfLines; j++) {

                yPos -= LINE_SPACING;
                connectorNode = nodeManager.create("N" + (+
                    +nodeCount));

                connectorNode.setNetPoint(sourceNode.getxCoordinate(),
                    yPos);
                sourceNodes.add(connectorNode);
            }

            sourceNode.setLocalConnectorNodes(sourceNodes);

            yPos = targetNode.getyCoordinate();

            // connector nodes at the destination are drawn
            for(j=0; j<numberOfLines; j++) {

                yPos -= LINE_SPACING;
                connectorNode = nodeManager.create("N" + (++nodeCount));

                connectorNode.setNetPoint(targetNode.getxCoordinate(),
                    yPos);
                targetNodes.add(connectorNode);
            }

            targetNode.setLocalConnectorNodes(targetNodes);

            // it is assumed the number of source Nodes is equal to the number of
            // target Nodes. The other scenarios are omitted for simplicity.
            for(j=0; j<sourceNodes.size(); j++) {

```

```

        node1 = sourceNodes.get(j);
        node2 = targetNodes.get(j);

        // the method creates the paths that form the section between nodes
        // node1 and node2, so that each path has constant gradient and speed limit.
        createSection(n1, n2);
    }

    fromNode = sectionDefinition.getNextNode();
}

}

}

/* Defining the entry, main and branch paths at points */
private void wirePoints() {

    nodes = nodeManager.getEntities();

    while(nodes.hasNext()) {

        node = nodes.next();

        if(node.isPoints()) {

            path1 = paths.getFirstPath();
            path2 = paths.getSecondPath();
            path3 = paths.getThirdPath();

            // getting the direction of the path on the display (in degrees)
            path1Angle = path1.getDisplayAngleRelativeTo(node);
            path2Angle = path2.getDisplayAngleRelativeTo(node);
            path3Angle = path3.getDisplayAngleRelativeTo(node);

            // measuring the angle differences between paths
            path1To2Diff = getAngleDifference(path1Angle, path2Angle);
            path2To3Diff = getAngleDifference(path2Angle, path3Angle);
            path3To1Diff = getAngleDifference(path3Angle, path1Angle);

            // comparing the angle differences to understand which path is the
            // entry/main/branch
            if(path1To2Diff < path2To3Diff && path1To2Diff < path3To1Diff) {
                // Path 3 is the entry
                node.setStart(path3.getOppositeNode(node));
                node.setEndOfMainTrack(path1.getOppositeNode(node));
                node.setEndOfBranchTrack(path2.getOppositeNode(node));

            } else if(path2To3Diff < path1To2Diff && path2To3Diff <
                path3To1Diff) {
                // Path 1 is the entry
                node.setStart(path1.getOppositeNode(node));
                node.setEndOfMainTrack(path2.getOppositeNode(node));
                node.setEndOfBranchTrack(path3.getOppositeNode(node));
            } else {
                // Path 2 is the entry
                node.setStart(path2.getOppositeNode(node));
            }
        }
    }
}

```



```

        node.setEndOfMainTrack(path1.getOppositeNode(node));
        node.setEndOfBranchTrack(path3.getOppositeNode(node));
    }
}

}

}

/* Creating the signals along a section
The case study required adding distance signals before a station (or passing
loop)
*/
private void createSignals(){

    paths = getStationPaths();

    for(Path path : paths) {

        branchesToLeft = getBranchesForStationWithId(path, path.getId(), LEFT_DIR);
        branchesToRight = getBranchesForStationWithId(path, path.getId(),
RIGHT_DIR);

        for(leftBranch : branchesToLeft){

            nextPath = leftBranch.getStartNode().getOppositePath(leftBranch);
            // a node is added at the default distance DIST_SIG_LENGTH and set to be
            a signal
            addDistantSignal(nextPath, DIST_SIG_LENGTH);
        }

        for(Path rightBranch : branchesToRight){

            nextPath = rightBranch.getStartNode().getOppositePath(rightBranch);
            addDistantSignal(nextPath, DIST_SIG_LENGTH);
        }
    }
}

/* Creation of the signalling routes */
private void createBlockSectionRoutes(nodeList, path, node) {

    paths = getPathList();

    while(paths.hasNext()) {

        path = paths.next();

        initialNode = path.getStartNode();
        initialPath = path;

        routeNodes.add(initialNode);

        nextPath = initialNode.getOppositePath(initialPath);
        nextNode = nextPath.getOppositeNode(initialNode);
    }
}

```

```

// if a signal is in the same direction of the start signal or it is a line terminator,
then the algorithm stops generating signals
if(!((nextNode.isSignal() && signalInSameDirection(nextNode)) ||
nextNode.isLineTerminator())){

    routeNodes.add(nextNode);

    // if a node is a switch, the method is called recursively on the main and
    the branch paths
    if(nextNode.isPoints() &&
nextNode.getEntryPath().getName().equals(nextPath.getName())){

        mainPath = nextNode.getEndOfMainPath();
        branchPath = nextNode.getEndOfBranchPath();

        // the method is recursive
        createBlockSectionRoute(routeNodes, mainPath,
mainPath.getOppositeNode(nextNode));

        // the method is recursive
        createBlockSectionRoute(routeNodes, branchPath,
branchPath.getOppositeNode(nextNode));

    }
    else{
        nextPath = nextNode.getOppositePath(nextPath);
        nextNode = nextPath.getOppositeNode(nextNode);

        // the method is recursive
        createBlockSectionRoute(routeNodes, nextPath, nextNode);
    }
} else {

    // the route is created as a list of nodes
    routeNodes.add(nextNode);
    route = routeManager.create(routeId);
    route.setNodes(routeNodes);

}

} // while
}

/* Train definition in terms of maximum speed, length, gross weight, maximum
acceleration and deceleration rate, tractive effort curve and resistance curve */

private void createTrains() {

    for(id : rollingStockDefinitions.list()){

        vehicle = rollingStockDefinitions.get(id);

```

```

vehicleType = vehicleTypeManager.create();

// setting the maximum speed, length and gross weight
vehicleType.setDescription("Class " + vehicle.getRollStockID());
vehicleType.setvMax(vehicle.getVelMax());
vehicleType.setLength(vehicle.getLength());
vehicleType.setWeight(vehicle.getGrossWeight());

    max_acceleration = MAX_TRACTIVE_EFFORT/vehicleType.getWeight();
vehicleType.setMaxAcceleration(max_acceleration);
vehicleType.setMAXDeceleration(max_acceleration - 0.1);

    // the train resistance curve is embedded in BRaVE
for(f=0; f< DEFAULT_RESISTANCE_CURVE.length; f++) {

        vehicleType.addRollingResistanceTableEntry(DEFAULT_RESISTANCE_CURVE[f][0], DEFAULT_RESISTANCE_CURVE[f][1]);
    }

    functionsList = vehicle.getTractiveEffortCoefficients();

    // the tractive effort curve is built using the coefficients provided in Viriato
    for(currentEntry : functionsList){

        fromVel = vehicleSpeed;
        toVel = currentEntry.getToVel();

        c0 = currentEntry.getC0();
        c1 = currentEntry.getC1();
        c2 = currentEntry.getC2();

        // the tractive effort is assumed to be parabolic
        tractiveEffort = c0 + c1* vehicleSpeed + c2 *
            Math.pow(vehicleSpeed, 2);

        vehicleType.addEntry(fromVel, toVel, tractiveEffort);

        vehicleSpeed = toVel;
    }

    // calculating the number of coaches based on the train length; a multiple
    unit is assumed
    numberOfCoaches = (int)
    Math.round((vehicle.getLength()/DEF_VEHICLE_LENGTH));
    vehicleType.setCarCount(numberOfCoaches);
}
}
}

/* Definition of a train route and schedule */

```

```

private void createServices() {
    for(id : trainMainDefinitions.list()){
        tmd = trainMainDefinitions.get(id);

        Service service = serviceManager.create(service_id);
        service.setTrainName(tmd.getTrainID());
        service.setTrainTypeName(tmd.getTrainID());
        service.set_class(tmd.getTrainTypeID());
        service.setTrainDescription(tmd.getTrainDescription());
        setOperatingDays(service, tmd.getOperatingID());

        trainRoute = tmd.getTrainRoute();

        // first node of the route is a station
        viriatoNode = trainRoute.getFirstNode();
        sectionTrack = viriatoNode.getSectionTrack();

        startNodeName = trainRoute.getNode().getName();

        stationPath = getPath(viriatoNode.getName());
        // looking for a route in the BRaVE database that includes the given station
        track
        routeName = findRoute(stationName, secTrack);

        beginNode = stationPath.getEndNode();
        beginNode.setTrackId(beginNode.getName());

        lastDeparture = tmd.getFirstDep();

        // first service entry
        serviceEntry = new ServiceEntry(routeName, ServiceEntry.TYPE_STOP);
        serviceEntry.setRequestedDeparture(lastDeparture);
        serviceEntry.setTrackId(beginNode.getTrackId());
        serviceEntry.setStationId(startNodeName);
        serviceEntry.setStopTime(DEF_STOP_TIME);
        service.addTimetableEntry(serviceEntry);

        // the code relative to intermediate stations has been omitted for simplicity

        // last node of a train route
        nextViriatoNode = trainRoute.get(trainRoute.size()-1);
        stationPath = getPath(nextViriatoNode.getName());

        stopNode = stationPath.getEndNode();
        stopNode.setTrackId(stopNode.getName());

        routeName = findRoute(stationName, secTrack);

        lastDeparture = lastDeparture + previousViriatoNode.getRuntime();

        // last service entry
        serviceEntry = new ServiceEntry(routeName, ServiceEntry.TYPE_STOP,
        lastDeparture);
        serviceEntry.setStationId(nextViriatoNode.getNode());
    }
}

```

```

        serviceEntry.setTrackId(stopNode.getTrackId());
        serviceEntry.setRequestedDeparture(lastDeparture);
        service.addTimetableEntry(serviceEntry);
    }

    /* Generating track circuits, route interlocking and signal behaviour */
    public void generateInterlocking() {
        generateTrackCircuits();
        generateRouteInterlockings();
        generateSignalScripts();
    }

```

```

%%%% Matlab code for interfacing BRaVE to Matlab evaluation framework

length_d_sim = zeros(1,size(f,1));
disturbed_sim_master = cell(max(length_d_sim),9,size(f,1));

%% Opening the simulation file produced by BRaVE (in csv format)
fh = fopen(filename, 'r');
chunksize = 1e3;
no_lines = 0;

%% Counting the number of lines in the file
while ~feof(fh)
    ch = fread(fh, chunksize, '*uchar');
    if isempty(ch)
        break;
    end
    no_lines = no_lines + sum(ch == sprintf('\n')); %\n is newline
end
fclose(fh);

%% Reading in data
fid = fopen(filename);
headers = textscan(fid, '%s', 4, 'delimiter', '\n');
data1 = cell(no_lines,13);
count = 0;

for k=1:no_lines-4
    Cline = fgetl(fid);
    if Cline(1) == 'A' || Cline(1) == 'D'
        count = count+1;
        C = textscan(Cline, '%s %s %s %s %s %s %s %s %s %s', 1,
            'delimiter', ',');
        data1(count,1:9) = C;
    elseif Cline(1) == 'O' % O
        count = count + 1;
        C = textscan(Cline, '%s %s %s %s %s %s %s %s %s %s', 1,
            'delimiter', ',');
        data1(count,1:9) = C;
    end
end
data1(count+1:end,:) = [];

%% Storing the data in a matrix
for i = 1:count
    d1(i,1) = data1{i,1};
    d2(i,1) = data1{i,2};
    d3(i,1) = data1{i,3};
    d4(i,1) = data1{i,4};
    d5(i,1) = data1{i,5};
    d6(i,1) = data1{i,6};
    d7(i,1) = data1{i,7};
    d8(i,1) = data1{i,8};
    d9(i,1) = data1{i,9};
end
data = [d1,d2,d3,d4,d5,d6,d7,d8,d9];

%% Closing the file
fclose(fid);
end

```



```

%% Extract of Matlab code for interfacing evaluation and optimisation
stages, both developed using Matlab
function importing_input_data_from_eval(filename)

%% Reading the XML file
try
    xDoc = xmlread(filename);
catch
    error('Failed to read XML file %s.',filename);
end

%% retrieving the root element of the file
allTrainData= xDoc.getElementsByTagName('trainData');
INPUT_DATA.number_of_trains = allTrainData.getLength;

%% Defining the structure that will contain the destination stations of all
the trains
INPUT_DATA.destination_loc = zeros(allTrainData.getLength, 1);

%% Additional parameters set manually
load('USER_INPUT.mat');
INPUT_DATA.total_journey_time_tolerance =
USER_INPUT.total_journey_time_tolerance;

%% Retrieving the data relative to a single train with id train_id
for i=0:allTrainData.getLength-1

    curr_train = allTrainData.item(i);
    train_id = char(curr_train.getAttribute('serviceId'));

    %% Destination stations
    INPUT_DATA.destination_loc(i+1,1) =
str2double(curr_train.getElementsByTagName('dest_location').item(0).getText
Content);

    %% Saving the data in the structure INPUT_DATA
    save 'INPUT_DATA.mat' INPUT_DATA;
end

```

```

%%%% Matlab code for interfacing Matlab to Viriato

function write_results_in_Viriato(database_name, train_ids,
train_arrival_times)

% Making a connection to the database using the ODBC driver.
conn = database(database_name);

% creation of a list containing the train names
list = [];
for v=1:size(train_ids,2)-1
    list = [list, train_ids{v}, '', ''];
end
    list = [list, train_ids{v+1}];

% Retrieving the current schedule for the trains in the list
where_clause = strcat(' WHERE `Train Definition`.TrainID IN ('',
list , '')');

curs = exec(conn, ['SELECT `Train Definition`.TrainID'...
    ' , `Train Definition`.RunTime'...
    ' , `Train Definition`.StopTime'...
    ' FROM `Train Definition` '...
    where_clause]);

curs = fetch(curs);

% train running times
service_runtimes = curs.Data;

% modify the train running times
    for i=1:size(train_ids,2)

        single_train_times = strcmp(service_runtimes.TrainID(:),
train_ids(i));

        stopTime = service_runtimes.StopTime(single_train_times);
        service_runtimes.RunTime(single_train_times)=
        round(arrival_times(2:end) - departure_times(1:end-1));

        runTime = service_runtimes.RunTime(single_train_times);
        % updating the train running times in the database
        for j=1: size(service_runtimes.RunTime(single_train_times),1)

            where_clause = strcat('where TrainID = ',
train_ids(i) , '' and Sequence = ', num2str(j));

            update(conn, '`Train Definition`','RunTime', runTime(j),
where_clause);

        end
    end
end

```