Ant Colony Optimisation for Dynamic and Dynamic Multi-objective Railway Rescheduling Problems

by

Jayne Eaton

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

School of Computer Science and Informatics

De Montfort University

2017

Declaration of Authorship

The content of this submission was undertaken in the School of Computer Science and Informatics, De Montfort University, and supervised by Prof. Shengxiang Yang and Dr. Mario Gongora during the period of registration. I hereby declare that the materials of this submission have not previously been published for a degree or diploma at any other university or institute. All the materials submitted for assessment are from my own research, except the reference work in any format by other authors, which are properly acknowledged in the content. Part of the research work presented in this submission has been published or has been submitted for publication in the following papers:

J. Eaton and S. Yang. Dynamic railway junction rescheduling using population based ant colony optimisation. *In Computational Intelligence (UKCI), 2014 14th UK Workshop on*, pages 1-8, Sept 2014.

J. Eaton, S. Yang, and M. Mavrovouniotis. Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays. *Soft Computing*, 20(8):2951-2966, August 2016.

J. Eaton and S. Yang. Railway platform reallocation after dynamic perturbations using ant colony optimisation. *In IEEE Symposium Series on Computational Intelligence, 2016 Proceedings of the* IEEE, 2016.

J. Eaton, S. Yang and M. Gongora. Ant Colony Optimisation for Simulated Dynamic Multi-objective Railway Junction Rescheduling. *IEEE Transactions on Intelligent Transportation Systems,* 2017 (in press)

Abstract

Recovering the timetable after a delay is essential to the smooth and efficient operation of the railways for both passengers and railway operators. Most current railway rescheduling research concentrates on static problems where all delays are known about in advance. However, due to the unpredictable nature of the railway system, it is possible that further unforeseen incidents could occur while the trains are running to the new rescheduled timetable. This will change the problem, making it a dynamic problem that changes over time. The aim of this work is to investigate the application of ant colony optimisation (ACO) to dynamic and dynamic multi-objective railway rescheduling problems. ACO is a promising approach for dynamic combinatorial optimisation problems as its inbuilt mechanisms allow it to adapt to the new environment while retaining potentially useful information from the previous environment. In addition, ACO is able to handle multi-objective problems by the addition of multiple colonies and/or multiple pheromone and heuristic matrices.

The contributions of this work are the development of a junction simulator to model unique dynamic and multi-objective railway rescheduling problems and an investigation into the application of ACO algorithms to solve those problems. A further contribution is the development of a unique two-colony ACO framework to solve the separate problems of platform reallocation and train resequencing at a UK railway station in dynamic delay scenarios.

Results showed that ACO can be effectively applied to the rescheduling of trains in both dynamic and dynamic multi-objective rescheduling problems. In the dynamic junction rescheduling problem ACO outperformed First Come First Served (FCFS), while in the dynamic multi-objective rescheduling problem ACO outperformed FCFS and Non-dominated Sorting Genetic Algorithm II (NSGA-II), a state-of-the-art multi-objective algorithm. When considering platform reallocation and rescheduling in dynamic environments, ACO outperformed Variable Neighbourhood Search (VNS), Tabu Search (TS) and running with no rescheduling algorithm. These results suggest that ACO shows promise for the rescheduling of trains in both dynamic and dynamic multi-objective environments.

Acknowledgments

I would like to thank my supervisors Prof. Shengxiang Yang and Dr. Mario Gongora for their help and advice with this project and Dr. Simon Coupland and Dr. Steve Ackland for their help and advice on the physics of train simulation. I would also like to thank Dr Dave Kirkwood at the University of Birmingham for sharing his expertise on train simulations and for supplying the power and resistance tables and the energy equation used in the Stenson Junction simulation.

Thank you to the other PhD students in my office without whom this journey would have been a much more desolate undertaking. In particular thank you to Shouyong Jiang for his limitless knowledge of so many things, to Conor Fahy for making me laugh, all the time, to Muhannod Younis for his limitless good humour and optimism and to Manal Alghieth for her gentle and kind camaraderie. Also thank you to Dr. Michalis Mavrounitis for his advice on ACO algorithms, he's moved on to pastures new but he is not forgotten. Thank you also to Dr. Ben Passow for taking time to talk to me when I was lost in the PhD wilderness.

Thank you to my kind and patient husband, Nick, for his calm belief that I could do this, for his stoical proof-reading of my journal papers and for his endless baking of comfort food. Thank you to my two supportive and enthusiastic sons who were proud that their mum was doing a PhD and were always there to offer encouragement and advice. Thank you also to my mum and my sister for their unwavering belief in me.

Finally thank you to my renal nurse Emma, my consultant Mr. Ferraro and my surgeon Mr. Dutta at Nottingham City Hospital, without whom I may have failed to survive to the end of this thesis. And thank you again to my husband who plans to give me one of the greatest gift of all, a new kidney and a chance of a normal life without dialysis.

Contents

List of Figures

List of Tables

Nomenclature

Algorithms

ACO	Ant Colony Optimization
ACS	Ant Colony System
ANN	Artificial Neural Network
BB	Branch and Bound
BC	Branch and Cut
BF	Brute Force
DE	Differential Evolution
DP	Dynamic Programming
DTBE	Decision-tree based elimination
FCFS	First Come First Served
FFP	First Free Platform
FIFO	First In First Out
FLFS	First Leave First Served
FSFS	First Scheduled First Served
GA	Genetic Algorithm
MMAS	Max-Min Ant System
NSGA	Non-dominated Sorting Genetic Algorithm
NSGA-II	Non-dominated Sorting Genetic Algorithm II

P-ACO	Population Based Ant Colony Optimization
SA	Simulated Annealing
SPEA2	The Strength Pareto Evolutionary Algorithm 2
TMS	Traffic Management System
TOE	Timetable order enforced
TS	Tabu Search
VNS	Variable Neighbourhood Search

Acronyms

ARS	Automatic Route Setting
ATP	Automatic Train Protection
bTSP	Bi-objective Travelling Salesman Problem
CBTC	Communication Based Train Control
CDR	Conflict Detection and Resolution Problem
DMOP	Dynamic Multi-objective Optimisation Problem
DOP	Dynamic Optimisation Problem
DRJRP	Dynamic Junction Railway Rescheduling Problem
DSRP	The Dynamic Station Recheduling Problem
DTSP	Dynamic Travelling Salesman Problem
HST	High Speed Train
ILP	Integer Linear Programming
IP	Integer Programming
JSP	Job Shop Scheduling Problem
MILP	Mixed Integer Linear Programming
MOA	Multi-objective Algorithm

MOACO	Multi-objective Ant Colony Optimisation
MOKP	Multi-objective Knapsack Problem
MOP	Multi-objective Problem
MRT	Mass Rapid Transit
POF	Pareto Optimal Front
POS	Pareto Optimal Set
rtRTMP	Real-time Railway Traffic Management Problem
SNCF	Société nationale des chemins de fer français
TSP	Travelling Salesman Problem

Symbols - ACO Algorithms

α	a constant which determines the relative influence of the pheromone
β	a constant which determines the relative influence of the heuristic
$\Delta\tau_{ij}^{best}$	pheromone update value in MMAS
η_{ij}	the heuristic information
ρ	the pheromone evaporation rate where $0 < \rho \leq 1$
τ_0	the initial pheromone value
τ_{ij}	the pheromone information

Glossary of Railway Terms

Name	Description
Automatic Block	A block system in which the state of each block section is monitored by automatic, rather than manual, devices.
Block	A section of track between two signals
Blocking Time	The total time a train occupies a block section. It includes its running time, the time it takes for the end of the train to leave the block section, the driver's sight distance to the signal and the time taken to release the occupied route.
Blocking Time Stairway	A time and distance graph of the blocking times for each block section on a train's route.
Dwell Time	The difference between the time a train stops at a station and the time it starts to leave.
Fixed Block System	A train movement safety system in which trains are prevented from entering a block section already occupied by another train.
Headway	The time interval between two consecutive trains
Interlocking	Consists of an arrangement of interlocked signals and points that are designed to ensure that a train will not proceed unless the route ahead is safe. Once a train's route is set the interlocking is implemented and all the relevant switches in the route are locked until the train has fully passed.
Macroscopic Models	Represent the railway network at a low level of detail. They usually ignore block sections and signals and use fixed train running and headway times .

Name	Description
Microscopic Models	Represent the railway network at a high level of detail taking into account block sections, signals and detailed train running and headway times.
Minimum Headway	The smallest safe time interval between two consecutive trains
Moving Block System	A system in which safe distances between trains are calculated and train speeds are adjusted to maintain those safe distances and prevent collisions.
Primary Delay	An initial delay caused by a disrupting incident such as signal failure, increased station dwell, train breakdowns etc.
Route	The path a train takes from its origin to its destination. It consists of an ordered sequence of timing points.
Running Time	The time a train takes to traverse a block section
Secondary Delay	'Knock-on' delays to other trains caused by conflict with the primary delayed train.
Timing Point	A timed location on a train's route that appears in the train schedule with a designated arrival and departure, or passing, time.
TIPLOC	A code that identifies a timing point.

Glossary of Algorithms

Name	Acronym	Description
Branch and Bound	BB	Represents the search space as a tree and travels from the top to the bottom of the tree to find candidate solutions. If a branch can be proved to be non-optimal, by comparison with an estimated upper bound, it is pruned.
Brute Force	BF	Involves enumerating all possible solutions. It will always find the optimal solution
Combinatorial optimisation		Involves finding values for discrete variables to obtain the optimal solution with respect to a given objective function.
CPLEX Optimizer	CPLEX	A mixed-integer programming tool which can solve linear problems using the simplex method with a Branch and Bound algorithm or Branch and Cut.
Differential Evolution	DE	DE is a population based search where individuals consist of vectors of real values. New individuals are generated by adding a weighted difference between two individuals to a third individual. If the new individual is superior to the comparison individual it replaces it.
Discrete Variable		A variable that can only take certain values from a finite set.
DP	Dynamic Programming	Involves breaking the problem down into stages and solving each stage separately.

Name	Acronym	Description
Decision-tree based elimination	DTBE	Works on a decision tree and reduces the search time by pruning the tree according to some heuristic rules.
Evolutionary Algorithm	EA	A broad term covering different types of evolutionary algorithms. For example, a GA is a class of evolutionary algorithm.
First Come First Served	FCFS	Trains pass through the junction in the order that they arrive at the junction.
First In First Out	FIFO	Assigns the block section to the first train to arrive.
First Leave First Served	FLFS	Precedence is given to the train that would be able to leave a block section first.
First Out First In	FOFI	Assigns the block section to the train calculated to leave the block section first.
First Scheduled First Served	FSFS	Precedence is given to the first train scheduled to use the block section.
Genetic Algorithm	GA	Evolves a solution by improving a population of candidate solutions. Uses mutation and crossover and selection operators.
Greedy Algorithm		Finds a solution by choosing the best option at each decision point.
Heuristic		Tends to involve the implementation of 'rules' for the algorithm to follow. It can produce fast, but often sub-optimal, solutions.
Q-learning		A reinforcement learning approach based on a single agent. The agent performs actions which are rewarded or punished. The agent's goal is to maximise its total reward.

Name	Acronym	Description
Simulated Annealing	SA	Based on a local search. At each step of the algorithm the current solution is compared to a solution close to the current solution and is replaced if the new solution is better. The comparison depends on a temperature parameter which decreases over time. At the beginning, temperature is high and the algorithm makes large movements around the search space but as the temperature converges towards zero the algorithm converges on a solution.
Metaheuristic		A high level algorithmic framework that can be used to define heuristic methods that are applicable to a wide range of different problems.
Stochastic Programming	SP	Mathematical programming that includes a stochastic element, for example a probability distribution. It is used when uncertainty is present. The scheduling model produced consists of a schedule for the first period and a set of decision rules (recourse decisions) that define which second period action should be taken in response to different scenarios.
Tabu Search	TS	Is based on a local search. At each step of the algorithm the current solution is compared to a solution close to the current solution. Local minima are avoided by using memory, once a potential solution has been encountered it is marked as 'tabu' and is not used again.

Name	Acronym	Description
Variable Neighbourhood Search	VNS	Is based on a local search on increasingly distant neighbourhoods. At each step of the algorithm the current solution is compared to a solution in the current neighbourhood, if the new solution is not an improvement the next neighbourhood is explored until all neighbourhoods have been investigated.

Chapter 1

Introduction

The problem of rescheduling trains after a delay is an important concern of the railway industry. Train timetables are designed to ensure the conflict-free running of the railway network; however, in the real world, delays may be caused by factors such as train failures, crew shortages, excessive dwell time at the station and obstructions on the line. A late arriving train will miss its scheduled time-slot and may cause 'knock-on' delays to other trains, resulting in the delay propagating throughout the network.

Delays are a common occurrence on the British railway network. According to Network Rail's Annual Report for 2016 [3] over 10% of trains failed to arrive on time, that is they failed to arrive within 10 min of scheduled arrival for long distance trains and 5 min of scheduled arrival for regional and London/South East trains. In addition, 3.1% of trains were cancelled or were classed as significantly late (over 29 minutes late).

Recovering the timetable after a delay is essential to the smooth and efficient operation of the railways for both passengers and railway operators. However, rescheduling trains is a challenging problem due to the involvement of usually conflicting multi-objectives (e.g., minimising delays, minimising broken train connections, and minimising costs), and dynamic and uncertain conditions (e.g., multiple consecutive train delays, ongoing train arrivals). At the present time, the dynamic nature of the train rescheduling problem is rarely considered in railway rescheduling research (see Chapter 3). Most train rescheduling problems are regarded as static in that all delays are known about in advance and no further unforeseen incidents occur while the original delay is being resolved. However, in the real world more unforeseen incidents may occur while trains are in the process of being rescheduled. In fact, the ON-TIME project report on the functional and technical requirements specification for large-scale perturbation management observes that in some inci-

dents secondary problems may become apparent after resolving the first incident which will result in the need for re-planning [4, p. 84]. If a rescheduling problem changes, due to unforeseen or secondary problems, it becomes a dynamic problem that changes over time.

1.1 Dynamic Railway Rescheduling

Researchers are starting to become aware of the need to address dynamic railway rescheduling problems. Meng and Zhou state that:

> "Continuing advances in real-time train scheduling algorithms essentially depend on modeling and algorithmic advances that recognize the dynamic and stochastic nature of the problem of interest."[5, p. 1100]

Corman *et al.* observe that:

> "...while online static rescheduling has reached a wide degree of dissemination, much is still to be done with regard to the probabilistic nature of the railway traffic rescheduling problems, and also how to best take uncertainty into account for future states.."[6, p. 1274]

While Cacchiani *et al.* [7, p. 34], in their overview of recovery models and algorithms for real-time railway rescheduling, point out the need to deal in an integrated way with the inherent uncertainty and dynamics in a real-time rescheduling environment.

It appears that there is an growing realisation of the need to consider the dynamic nature of the railway rescheduling problem, however, at the present time little work has addressed the issue directly. The goal of this work is to contribute towards the field of dynamic railway rescheduling by investigating the ability of ant colony optimisation (ACO) algorithms to solve such problems.

1.2 What are Dynamic Optimisation Problems (DOPs)?

DOPs are problems where an aspect of the problem changes over time, resulting in the optimal solution(s) to the problem also changing over time.

Nguyen [8] defined a DOP as follows:

> "Given a dynamic problem $f(t)$, an optimization algorithm G to solve $f(t)$, and a given optimization period t_{begin}, t_{end}, $f(t)$ is called a dynamic

optimization problem in the period t_{begin}, t_{end} if during t_{begin}, t_{end} the underlying fitness landscape that G uses to represent $f(t)$ changes and G has to react to this change by providing new optimal solutions."

Hence a problem can only be defined as a DOP if the algorithm has to react to the change to produce new optimal solutions. The railway rescheduling problems addressed in this work fall within this definition as in each case the algorithm attempts to find a new optimal solution after a change in the problem.

The fact that evolutionary computation (EC) techniques, such as ACO, are inspired by processes that occur in nature, processes that continually adapt to the changing environment suggest that EC techniques may be very applicable to dynamic problems.

Two important characteristics of a dynamic environment, as defined by Branke [9], are:

- The Frequency of Change - how often the environment changes.
- The Severity of Change - the magnitude of the change

In this work one of the aims is to investigate the effect the magnitude and frequency of the delay has on the performance of the algorithms. This approach is supported by Samà *et al.* [10, p15] who point out that there is a need to investigate the effect that the type of disturbance has on the algorithmic performance.

An important factor to consider in DOPs is the visibility of the change [9]. It may, in some problems, be difficult to determine if a change has taken place. However, this is not the case in railway rescheduling problems as changes are very visible. Railway dispatchers are provided with real-time information on the position of trains and train delays become apparent very quickly. The fact that changes can be easily observed removes the need for complex mechanisms to detect the change.

Another important issue with DOPs is the influence the solution chosen to be implemented before a change has on the environment after a change. For example in railway rescheduling the solution chosen will be partially implemented before the next change in the environment which restricts the choice of future actions for the algorithm. Such a problem is known as a 'time-linked' problem [11] and, although it is recognised that the dynamic problems investigated in this thesis may have a time-linked nature, a thorough investigation of its impact is outside the scope of this thesis and is an avenue for future investigation.

1.3 Dynamic Multi-objective Railway Rescheduling

A second goal of this work is to consider not only dynamic, but dynamic multi-objective railway rescheduling problems. Although there has been some previous research on multi-objective railway rescheduling, for example [12, 13, 14, 15], most researchers combine the objectives into a single weighted objective and solve the problem as a single objective problem. There are two disadvantages to this method; the first is that the weights have to be determined in advance using domain knowledge; the second is that it assumes the relative importance of each objective does not change over time. This may not always be the case, for example the objectives in the early morning rush hour may differ from those in the mid-afternoon when the rush has died down.

A more flexible approach would be to produce a set of 'trade-off' solutions, known as a Pareto optimal front (POF), which would allow a dispatcher to choose the best solution to suit the current situations. Corman *et al.* [16] is one of the few researchers to have produced a Pareto optimal front of 'trade-off' solutions for a railway rescheduling problem with the multiple objectives of minimising delay and maximising retained train connections. This allows dispatchers to choose between solutions that have either a low delay but higher numbers of missed connections, or have a high delay with lower numbers of missed connections. This would allow the dispatcher more flexibility in their decision making and enable them to make decision that are relevant to the current situation. The difference between this work and the work presented in this thesis is that the problem investigated by Corman *et al.* was a static multi-objective problem.

Very few researchers consider railway rescheduling problems that are both dynamic and multi-objective (DMOP). The work by Fernández-Rodríguez *et al.* [17] is unusual in that it produces a POF for a multi-objective railway rescheduling problem and also considers the dynamic nature of the problem. However, it optimised the speed-profile for only one train and did not consider multiple trains. In contrast, in this work, the problems under investigation involve multiple trains.

The exploration of multi-objective, dynamic railway rescheduling problems involving multiple trains appears to be a little investigated area at the present time and reveals a gap in current railway research. Therefore, a second goal of this work is to contribute towards the field of dynamic multi-objective railway rescheduling by investigating the ability of ant colony optimisation (ACO) algorithms to solve such problems.

1.4 Motivation

The motivation behind this work is to contribute towards the development of real-time rescheduling algorithms that can be used in a computerised system, by a train dispatcher, to aid decision making.

In a real world rescheduling situation, the problem complexity, time constraints and limited decision support may lead to sub-optimal dispatching solutions [18, p370]. In fact, Kecman *et al.* [19] observed that current rescheduling practice mainly involves the use of predetermined rules and relies heavily on the experience and skills of the personnel. Hansen [20] pointed out that, due to the short amount of time available to make rescheduling decisions, train dispatchers are often only able to perform only a few timetable modifications, such as adjusting train routes, orders and speeds and that often the dispatcher is unable to know the effectiveness of the adjustments they make. In addition, they often do not have enough time to compare the performance of alternative solutions meaning that, in practice, dispatching decisions are often sub-optimal [21]. It is apparent, therefore, that the rescheduling decisions made by a dispatcher may be enhanced by the development of computerised rescheduling systems.

As railway rescheduling problems may be both dynamic and multi-objective, making a contribution towards solving dynamic and dynamic multi-objective rescheduling problems is an important step forwards in the search for algorithms to incorporate into real-time rescheduling software that can be used by a dispatcher to help resolve delays and to improve the punctuality of the railway network.

1.5 Why Ant Colony Optimization (ACO)?

ACO is an optimisation algorithm based on the natural behaviour of ants (see Section 2.2.1). It is inspired by the ability of ants to follow pheromone trails laid down by other ants to discover food [22].

ACO was chosen for this study for a number of reasons. The first reason is that the problems addressed in this thesis are combinatorial optimisation problems and ACO was designed for combinatorial optimisation problems [22]. ACO has previously been applied to static railway rescheduling problems [2, 23] with good results. It has also been successfully applied to other DOPs [24, 25, 26, 27, 28]. This suggests that ACO may be applicable to dynamic railway rescheduling problems.

A second reason for choosing ACO is that it has inbuilt mechanisms to cope with dynamic changes, either by the transfer of pheromone trails from one change to the next or by the use of a memory to retain useful information from before the change to

the next dynamic environment (see Sections 2.2.3 and 2.2.2). In addition, ACO has the ability to cope with multi-objective problems due to its flexibility in being able to add multiple colonies, or multiple pheromone and heuristic matrices, to address the separate objectives. Further, with regards to multi-objective optimisation, the population-based nature of ACO means that multiple 'trade-off' solutions can be generated in one run of the algorithm, in contrast, classical optimisation methods may have to run the algorithm separately for each objective [29].

A third reason for choosing ACO is that it allows the problem to be constructed in a way that ensures that only feasible solutions can be produced. This can save computational effort that might be spent on weeding out infeasible solutions. In such a time-critical environment as railway rescheduling, good solutions need to be produced as fast as possible.

A fourth reason for focusing on ACO is that it is relatively simple to deal with problem constraints, such as restrictions on the platforms trains can use, by limiting the choices the ants can make without affecting the basic operation of the algorithm.

Finally, ACO has the advantage that it is easily adaptable to problems with different track topologies and numbers of trains.

Other algorithms, such as genetic algorithms (GAs), were considered for these problems but were discarded because they are designed for continuous problems. In addition, in the case of GAs, the inherent procedures of crossover and mutation could result in infeasible train resequencing solutions.

1.6 Aims

There are three main aims of this work:

1. To investigate the application of ACO to dynamic railway rescheduling problems, that is problems where not all the information about train delays or train movements are known about in advance and where further changes will occur while trains are in the process of being rescheduled.

2. To investigate the application of ACO to dynamic multi-objective railway rescheduling problems by introducing a second objective, that of minimising additional energy consumption.

3. To investigate the application of ACO to the resolution of delays at a UK railway station in a dynamic environment and to create a framework that allows two colonies of ants to work together to solve the problem. In this case the aim is to consider a larger area of the railway system by taking into

account the effect the local decisions made at the station have on the trains' ongoing journeys.

1.7 Unique Contribution

The goal of this thesis is to make contributions in three areas of railway rescheduling research; dynamic railway rescheduling; dynamic multi-objective railway rescheduling; and dynamic platform reallocation and rescheduling.

Dynamic railway rescheduling problems are rarely considered in current railway rescheduling research. The contributions of this work in this area include: the creation of a benchmark problem and simulator to investigate dynamic railway rescheduling problems; a contribution to the field of understanding of how ACO algorithms can be applied to dynamic railway rescheduling problems; and a contribution to the field of understanding of the effect that the characteristics of the delay, in terms of magnitude and frequency, have on the ability of the algorithms to solve dynamic rescheduling problems. The use of ACO algorithms for railway rescheduling problems is a little explored area. Current work using ACO for railway rescheduling, for example Fan *et al.* [2] and Samà *et al.* [23], apply it only to static rescheduling problems.

With regards to dynamic multi-objective railway rescheduling, previous works consider only static or multi-objective problems. They fail to take into account the dynamic and multi-objective nature of railway scheduling problems. The investigation of such a problem is a new contribution to railway rescheduling. The contributions of this work in this area include: the creation of a benchmark problem and simulator to investigate dynamic multi-objective railway rescheduling problems; an investigation into a railway rescheduling problem that is both dynamic and multi-objective; a contribution to the field of understanding of how ACO algorithms can be applied to railway rescheduling DMOPs; and an attempt to identify both the features of the algorithms necessary for good performance on this DMOP and also the effect of the frequency and magnitude of change on each algorithm's performance.

Dynamic platform reallocation and rescheduling refers to the two separate processes of reallocating trains to new platforms after a train delay and deciding the order the trains should leave the station in order to minimise the impact of the delay. The contributions of this work in this area include: the creation of a dynamic benchmark problem for a dynamic station rescheduling problem based on Network Rail's schedule feed [30]; a contribution to the field of understanding of how ACO algorithms can be applied to the field of dynamic railway rescheduling,

specifically dynamic platform reallocation and resequencing; and the creation of a unique framework that combines two colonies of ants, one that reallocates trains to platforms after a delay and the other that determines the order the trains leave the station.

1.8 Thesis Structure

The remainder of this thesis is organised as follows:

Chapter 2 explains the railway terms used in this thesis and describes the ACO algorithms used to address the railway rescheduling problems considered in this work.

Chapter 3 discusses previous work in the area of railway rescheduling for both static, dynamic and multi-objective problems.

Chapter 4 describes a study of a dynamic junction rescheduling problem and the simulator used to model it. The problem is based on Stenson junction on the British railway network. It describes the results of applying a number of ACO algorithms to the problem and the use of immigrants to repair the ACO solutions after a change to make them suitable for the next dynamic change. Immigrants are an EC term to describe new solutions that are created and introduced into the algorithm while it is running.

Chapter 5 describes an extension of the above dynamic problem to introduce a second objective of minimising additional energy consumption. This makes the simulator a dynamic multi-objective railway rescheduling simulator. Various ACO algorithms were applied to the problem and were compared with NSGA-II, a 'state-of-the-art' DMOP algorithm and FCFS.

Chapter 6 takes a different approach to railway rescheduling by extending the problem to a station and its surrounding area. The motivation behind this is to assess the impact of the decisions made at the station on the ongoing journey of each train. In this case the model was created from the schedule feed based on Network Rail's Integrated Train Planning System (ITPS) [30] and is made dynamic by the addition of different magnitudes of delays at different time intervals (frequencies). The problem is broken down into two sub-problems. The first sub-problem is that of reallocating trains to new platforms, the second is that of deciding the order the trains leave the station (resequencing). Each sub-problem is addressed with a different colony of ants and a unique framework is presented that combines the two colonies to produce solutions for the combined station reallocation and rescheduling problem.

Finally Chapter 7 concludes the thesis by summarising the results and contributions of this work. It also includes a discussion of the wider application of this work and ideas for future investigations.

Chapter 2

Of Ants and Trains

The aim of this chapter is twofold. The first aim is to describe the ideas and terminology that form the basis for the train simulations and models created in this work. The second aim is to give an overview of the ACO algorithms that were used in these investigations. The chapter starts with an overview of railway technology followed by an introduction to ACO and a detailed description of the ACO algorithms.

2.1 Railway Terms

The British railway system has been in operation since the early nineteenth century and has resulted in a body of terminology that specifically describes railway operations and infrastructure. The following sections describe the railway terms that are referred to within this thesis when describing railway operations. A glossary of these terms can be found at the beginning of this thesis on page xvi.

2.1.1 Block Sections

In the railway network lines are sectioned into track segments. Track segments may be delineated by signals. These track segments are referred to as block sections and, in a fixed block system, are used to ensure the safe occupation of railway lines.

Each block section can only be occupied by one train at a time [31] and the signals ensure that if a block section is occupied incoming trains are forced to stop before entering that section. The length of the block sections affects the capacity of the line, the longer the block sections the smaller the capacity.

2.1.2 Blocking Time

The blocking time is the time interval in which a block section is exclusively occupied by a train. However, the blocking time is often longer than the amount of time the train physically occupies the track section as it includes other factors such as:

- The time the driver takes to see the signals (sight-distance).
- The time to release the occupied route.
- The clearing time for the tail of the train (last axle) to clear the block section.

If all the blocking times for each of the block sections the train traverses on its journey are plotted in a graph they produce a 'blocking-time stairway'. An overlap between the blocking times of different trains reveals a timetable conflict [20].

2.1.3 Running Time

The running time of a train is the time a train takes to traverse a block section [32]. It starts when the head (first axle) of the train enters the block section and ends when the head of the train reaches the end of the block section [33].

2.1.4 Dwell Time

Dwell time is the time between a train stopping at a station platform and beginning to leave that same platform.

2.1.5 Headway

Headway is the time interval between two consecutive trains. The minimum headway is the smallest safe time interval between two consecutive trains and is the distance between the blocking times of two consecutive trains on a block section [34]

2.1.6 Fixed Block Systems

In a fixed block system a train may not generally enter a block section until the train ahead has cleared it [34]. This ensures collision free traversal of the railway lines. A drawback of a fixed block system is that long block sections are needed to accommodate fast trains, due to the increased stopping distance, and this reduces the capacity of the line. An alternative to a fixed block system is a moving block system.

2.1.7 Moving Block System

In a moving block system block sections are considered to have a length of zero [34] and instead safe distances between trains are calculated and train speeds are adjusted to maintain these safe distances [35]. This system requires the position, speed and direction of the trains to be known at all times. Drivers do not make use of line-side signals but respond to instructions passed to them directly from a control centre. Moving block systems have the advantage that trains can be run closer together which increases the capacity of the line. However, the increase in capacity is dependent on the speed differences between consecutive trains. It works best on railways where the trains all run at the same speed, for example metro systems and mass transit lines. On lines where there are trains running at different speeds the capacity improvement is limited [34].

At the present time moving block is used in only a few areas of the British railway network, such as parts of the London underground and London's Dockland Light Railway. Pachl [34] observed that fixed block is the most commonly used system in today's railways. For these reasons the simulations and models used in this work simulate fixed block, rather than moving block, technology.

2.1.8 Automatic Block

When a train system uses automatic block technology the state of the block sections, in terms of occupation and clearance, is monitored by automatic devices. This removes the need for manual operators to check the trains have cleared a track section. The automatic devices may consist of either track circuits or axle counters. In a track circuit an electric current is passed to a detection device at the end of the track. When the axles of a train move onto a track section they short circuit the current which shuts off the flow to the detection device and indicates that the track section is occupied. In contrast, axle counters work by counting axles and comparing the number of axles that enter the track section to the number that leave.

2.1.9 Interlocking

Interlocking prevents trains from entering a junction or a crossing unless it is safe to do so. It consists of an arrangement of interlocked signals and points [34] that are designed to ensure that a train will not proceed unless the route ahead is safe and there is no possibility of it coming into conflict with any other train. Once a train's route is set the interlocking is implemented and all the relevant switches in the route are locked until the train has fully passed.

2.1.10 Timing Points

A timing point is a timed location on a train's route that appears in the train schedule with a designated arrival and departure, or passing, time.

2.1.11 TIPLOC

A TIPLOC (Timing Point Location), is a code that identifies a timing point. For example the TIPLOC code for Leicester station is LESTER.

2.1.12 Route

A train's route is the path it takes from its origin to its destination. It consists of the ordered sequence of timing points it passes in order to reach its destination.

2.1.13 Primary Delay

A primary delay is an initial delay, in a railway rescheduling problem, caused by a disrupting incident such as signal failure, increased station dwell, train breakdowns etc.

2.1.14 Secondary Delay

Secondary delays are 'knock-on' delays to other trains caused by conflict with the primary delayed train.

2.1.15 Microscopic Models

Microscopic models consider train movements at a high level of detail. They take into account individual block sections, signals and detailed running and headway times [7]. Due to computational overheads microscopic models are, at this present time, generally only able to model small areas, up to 50km, and relatively small time horizons (less than 1 hour) [6, p1275].

2.1.16 Macroscopic Models

Macroscopic models represent the railway network in less detail than microscopic models. They usually represent the topology as a series of links connecting stations and do not take into account block sections or signals. In addition fixed running times and headways are often used [18].

In the following section the concepts behind ACO are described in detail in order to provide a background to the work completed in this thesis.

2.2 Ant Colony Optimisation (ACO)

ACO is a metaheuristic algorithm often applied to combinatorial optimisation problems. A metaheuristic is a high level algorithmic framework that can be used to define heuristic methods that are applicable to a wide range of different problems. Combinatorial optimisation problems involve finding values for discrete variables to obtain the optimal solution with respect to a given objective function. A discrete variable is a variable that can only take certain values from a finite set. The railway rescheduling problems in this thesis are combinatorial optimisation problems.

A combinatorial optimisation problem can be defined as a triple (S, f, Ω), where S is a set of candidate solutions, f is the objective function and Ω is a set of constraints. A solution s in the feasible solution set $\tilde{S}$ ($\tilde{S} \subseteq S$) is a solution that satisfies all the constraints in Ω. f assigns an objective value, $f(s)$, to each s in the set of candidate solutions ($s \in S$). The aim is to find the globally optimally feasible solution, $s*$ ($s* \in \tilde{S}$) that gives the best objective value in terms of the objective function. The work in this thesis concentrates on minimisation problems where the best solution is the solution with the lowest cost in terms of the objective function, such that $f(s*) \leq f(s)$ for all $s \in \tilde{S}$.

2.2.1 The Basic ACO Algorithm

As described in [1], ACO is an optimisation algorithm inspired by the ability of ants to follow pheromone trails laid down by other ants to discover food [22]. As ants move backwards and forwards from the nest to a food source they lay down pheromones on the ground which can be sensed by other ants. Ants choosing the shortest path to the food source will return more quickly which ensures that the shortest path accumulates more pheromone. Ants tend to probabilistically choose paths with the strongest pheromone concentration which means that a path with high pheromone levels will attract more ants and accumulate even more pheromone. In this way, the shortest path to a food source is marked by the strongest pheromone trail. However, if this trail were to persist after the food source was depleted, it would seriously hamper the ants' ability to find food. Therefore, pheromone trails evaporate over time to allow old decisions to be forgotten.

Figure 2.1 illustrates this principle. In Figure 2.1(a), ants search for food. The ant choosing the shortest path to the food will return quicker and lay down more

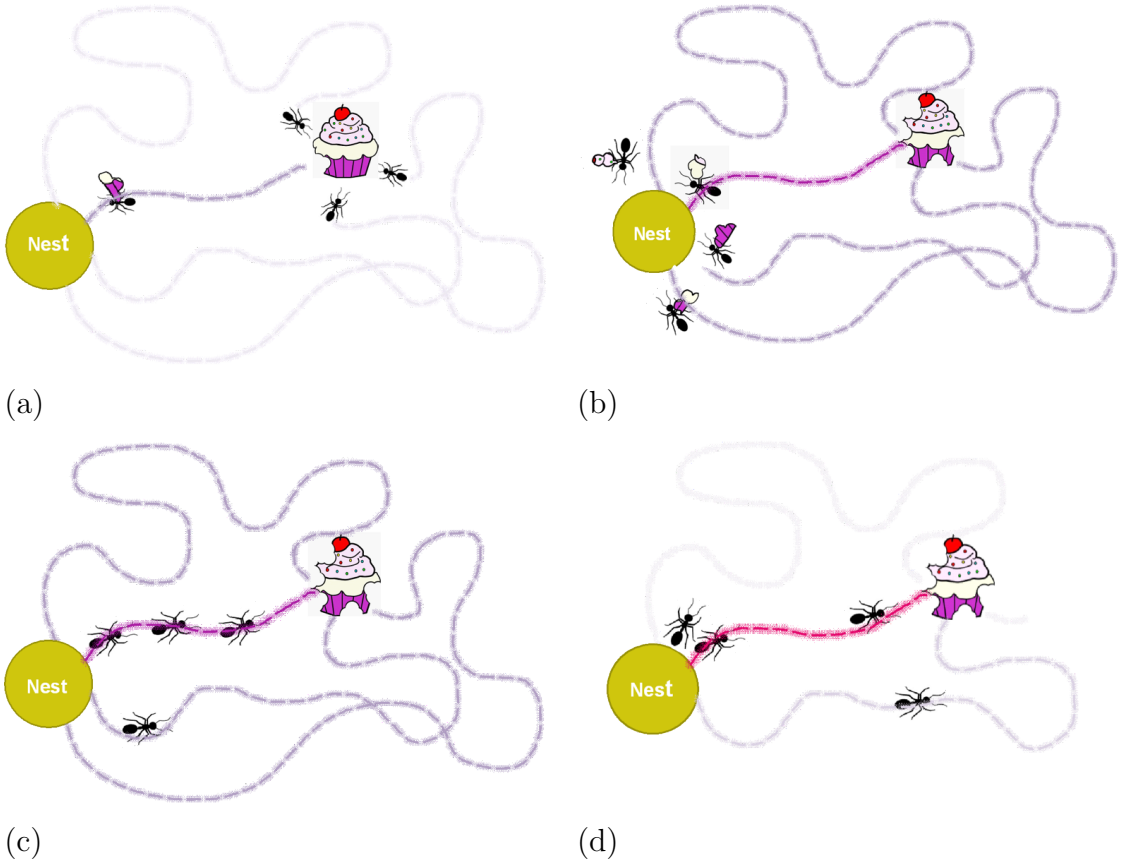


Figure 2.1: The principle behind ACO

pheromone thus reinforcing the shortest path. By the time the rest of the ants have returned to the nest the ant using the shortest path has fetched food again and laid down more pheromone (Figure 2.1(b)). When the ants leave in search of food again, they are more likely to follow the path with the highest pheromone level which reinforces the path even more (Figure 2.1(c)). Over time old, unreinforced pheromone trails evaporate so that old solutions can be forgotten (Figure 2.1(d)).

To apply ACO to an optimisation problem, the problem has to first be decomposed into a fully connected weighted graph $G = (V, E)$, where V is a set of vertexes or nodes, and E is a set of edges or connections between the nodes. The ants move along the edges of the graph from node to node recording the nodes visited. This list of visited nodes, sometimes called the ant's tour, is one possible solution to the optimisation problem. Pheromones are deposited on the edges of the graph by the ants according to how good an ant's solution is in terms of the optimisation objective. On the next iteration, the updated pheromone levels help to guide the ants to choose better nodes. Pheromones can be decreased as well as increased to model the process of evaporation which allows previous bad decisions to be forgotten. In

addition to the pheromone the edges may also be associated with a heuristic value, which is based on problem specific knowledge and provides additional guidance to the ants.

An ant, say ant k, when at node i, chooses the next node j in its neighbourhood N_i^k, probabilistically as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} \quad \text{if} \quad j \in N_i^k \tag{2.1}$$

where τ_{ij} is the pheromone information and η_{ij} is the heuristic information, α and β are constants which determine the relative influence of the pheromone and the heuristic values respectively. An ant chooses the next node in this way with a probability of $1 - q_0$; otherwise, it chooses the next best node in terms of the pheromone and heuristic values.

2.2.2 Population-Based ACO (P-ACO)

As described in [1], the above algorithm, however, does not provide any mechanism for allowing the ants to adapt to a change in the environment. Once the ants have converged on a solution, the resulting loss in diversity will make it difficult for them to adapt to a change in the problem and, in addition, the pheromone trails laid down for the previous environment may not provide any useful guidance to the ants in the new environment [26]. One option is to restart the algorithm after a change but such an action is not only computationally wasteful but also results in the loss of information that has the potential to be useful in the new environment.

To address this problem, Guntsch and Middendorf [36] introduced a Population based ACO (P-ACO) algorithm. In this algorithm, the best ant found at each iteration is stored in a memory, called the population-list, and only the ants in this list are used to update the pheromone levels. When the population-list reaches its designated limit, an ant is removed and the pheromone trail for that ant is negatively updated. This provides a mechanism for allowing previous bad decisions to be forgotten. To prevent the pheromone levels from building up to a level which means that all ants follow the same path, the amount of pheromone on each edge is bounded between a minimum value and a maximum value.

This memory of best iteration ants means that solutions made before a change can be retained to provide valuable information for the new environment. However, to make the ants suitable for the new environment, they may have to undergo a repair operation. Once repaired, the pheromone information for the new environment can be computed from the tours of the fittest ants created before the change, thus

ensuring that information from the previous environment can be passed over into the new environment. Guntsch and Middendorf [36] found P-ACO to perform better than restarting the algorithm when the environment change was small and frequent and comparable with restart when the change was large and slow.

2.2.3 Max-Min Ant System (MMAS)

One of the most popular ACO algorithms is the Max-Min Ant System (MMAS) [37]. As described in [38], in this algorithm, all of the pheromone trails are initialised to a maximum value. After each iteration, all pheromone trails are evaporated as in Eq. (5.9).

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij},\ \forall(i,j) \in L, \tag{2.2}$$

where $L = E$ is the set of all pheromones and $0 < \rho \leq 1$ is the pheromone evaporation rate [22], which is a constant parameter of the algorithm.

After each iteration, the pheromone trails are updated to correspond to the tour T^{best} of either the best-so-far ant or the best iteration ant as in Eq. (2.3).

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best},\ \forall(i,j) \in T^{best}, \tag{2.3}$$

The update value $\Delta\tau_{ij}^{best}$ is $\frac{1}{S}$, where S is the fitness of the best ant. In a minimization problem, as the fitness of the ant improves, the value of the pheromone update increases correspondingly.

In MMAS, an ant chooses the next node as in Eq. (2.1). The pheromone trails in MMAS are bounded between a minimum τ_{min} and a maximum τ_{max} value. The reason for this is to counteract the increased possibility of stagnation that may occur as a result of allowing only the best ant to deposit pheromone. In addition, stagnation is addressed by reinitialising all trails to τ_{max} when the algorithm shows stagnation behaviour or there has been no change in the best fitness for a set number of iterations. MMAS is unusual in that all pheromone trails are initialised to the maximum value, this together with a small evaporation rate increases the exploration of the search space at the start of the search [37].

The pheromone bounds are given as follows: $\tau_{max} = \frac{1}{S}$ and $\tau_{min} = \frac{\tau_{max}}{a}$. Where S is the fitness of the best ant and a is a parameter of the algorithm. Each time a new best ant is found, the values for τ_{min} and τ_{max} are updated. In a minimization problem, this means that as the fitness of the best ant, i.e, S, improves the values for both τ_{min} and τ_{max} increase.

2.2.4 Ant Colony System (ACS)

ACS was developed by Dorigo and Gambardella [39, 40] as a solution to the TSP problem. In ACS, an ant makes a decision as to which node to choose next using the *pseudorandom proportional* rule given in Eq. (2.4). According to the value of q, which is a random variable uniformly distributed in $[0, 1]$, an ant k on node i chooses the next node j to be the node with the highest combined pheromone and heuristic value; otherwise, it chooses according to the probability distribution given in Eq. (2.1).

$$j = \begin{cases} argmax_{l \in N_i^k}\{\tau_{il}[\eta_{il}]^\beta\}, & \text{if} \quad q \leq q_0 \\ J & \text{otherwise} \end{cases} \tag{2.4}$$

The effect of this rule is that with probability q_0 ($0 \leq q_0 \leq 1$) the ant exploits the learned knowledge of the colony; otherwise, with probability $(1 - q_0)$, it performs a biased exploration of the search space where it is biased towards choosing the node with the best pheromone and heuristic values but may not necessarily do so.

In ACS, two types of pheromone trail updates take place: global pheromone update and local pheromone update.

Global Pheromone Trail Update

The global pheromone trail update takes place after all the ants have made their solutions. Its purpose is to reward the edges belonging to the best tour. In this phase, the best-so-far ant updates the pheromone trails on all of the edges in its tour as in Eq. (2.5), where ρ ($0 \leq \rho \leq 1$) is a pheromone decay parameter. The update incorporates both evaporation $(1-\rho)\tau_{ij}$ and pheromone deposit $\rho\Delta\tau_{ij}^{bs}$, where $\rho\Delta\tau_{ij}^{bs}$ is $1/S^{bs}$ and S^{bs} is the fitness of the best-so-far solution T^{bs}. The better the fitness of the solution, the more pheromone is deposited. The outcome of the formula is that the new pheromone value is a weighted average of the old pheromone value and the amount of pheromone deposited [22].

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs}, \ \forall (i, j) \in T^{bs}, \tag{2.5}$$

Local Pheromone Trail Update

The purpose of the local pheromone trail update is to encourage exploration. It is performed by every ant immediately after adding a node to its tour. The pheromone update is performed using Eq. (2.6), where ξ ($0 < \xi < 1$) and τ_0 are two parameters of the algorithm. The value of τ_0 is the initial value for the pheromone trail, which is set to be $1/nS^{nn}$, where n is the number of nodes and S^{nn} is the fitness of a tour

made by always choosing the nearest node in terms of the heuristic. Its effect is to reduce pheromone concentration along the edge that the ant has just travelled, making the edge less desirable to the following ant which encourages it to explore other edges and discourages stagnation where all ants follow the same trail. For this to work, all ants must move in parallel one step at a time.

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0, \tag{2.6}$$

In ACS pheromone trail limits are not set explicitly but are implicit within the update formulas in Eqs. (2.5) and (2.6). They can never drop below τ_0 because both update formulas always add an amount of pheromone greater than τ_0. In addition they can never rise above $\rho\Delta\tau_{ij}^{bs}$ [22].

2.2.5 ACO for DOPs

After a dynamic change the optimal solution may have changed. There are two approaches to dealing with this. The first is to restart the algorithm, but this will lead to the loss of potentially useful information and may increase the time taken to find a new optimal solution. The second approach is to carry information from the old environment to the new environment. This information may potentially be very useful in guiding the ants to find an optimal solution in the new environment and may speed up the search process saving time and computational effort.

ACO is very applicable to DOPs because it has the ability to retain useful information from one change period to the next. This information is encoded in the pheromone trails and can be thought of as the ants' 'memory' of previous good solutions. If the information is no longer useful it will evaporate over time and will be removed.

In addition one variation of the ACO algorithm, P-ACO actually holds a memory of good ant solutions that can be used to initialise the pheromone trails after a change to provide useful information in the new environment.

2.3 Summary

The aim of this chapter was to describe the ideas and terminology that form the basis for the train simulations and models created in this work and to give an overview of the ACO algorithms that were used to investigate these problems.

In the next chapter, previous work in the area of railway rescheduling is described and discussed in order to provide a context for the work presented in this thesis.

Chapter 3

Literature Review

In this chapter the recent literature on rescheduling trains in perturbed situations is presented and discussed. Railway rescheduling is a popular research area. This thesis is an exploration of the application of the EC technique of ACO to rescheduling problems, however, much of the existing work uses non-evolutionary techniques, such as Branch and Bound (BB), dynamic programming (DP) and Branch and Cut (BC). Many exact methods, such as modelling the problem as a Mixed Integer Linear programme (MILP) and solving with CPLEX are usually able to find the optimal solution but in a time-frame that is unacceptable in a real-world rescheduling scenario. For example, Semet and Schoenauer [41] found that a CPLEX optimiser took several hours to find a solution to a rescheduling problem on a section of the French railway. This suggests that EC techniques that can provide an optimal, or near optimal solution, within a realistic time frame have a part to play in railway rescheduling.

3.1 Railway Rescheduling Definitions

Within the literature there are a number of different definitions of the rescheduling problem. Corman *et al.* [42] and Dariano *et al.* [43] refer to it as the conflict detection and resolution (CDR) problem and define it as the problem of minimising timetable deviation by rescheduling train movements while ensuring that all train movements obey signal constraints and each train has a feasible speed profile. Samà *et al.* [10] describe the CDR problem as that of minimising the delay of all trains in a network by computing a deadlock and conflict-free rescheduling solution.

Sama *et al.* [23] and Pellegrini *et al.* [44] refer to railway rescheduling as the real time railway traffic management problem (rtRTMP) and define it as "the detection and resolution of conflicting requests in disturbed operations" [23]. While Corman

et al. [6, 1275] describe rescheduling as "the process of updating the timetable (the published plan of train departures, passing times and train arrivals, and routes over rail network), by taking into account the current position and speed of trains, and their delays."

In this work railway rescheduling is taken to be the problem of adjusting the scheduled routes and times of one, or several trains, in order to mitigate the effects of a delay.

3.2 The difference between Scheduling and Rescheduling

Only rescheduling research is considered here. Scheduling research, although similar in that it also assigns routes and arrival and departure times to trains, is not considered because of the underlying differences between scheduling and rescheduling. One of the main differences is that rescheduling has to take place in a very restricted time window. When a disruption occurs a new schedule has to be found as quickly as possible in order to minimise the effects of the delay, whereas scheduling can take many months to create a suitable timetable and is not designed to respond to real time changes.

In addition, rescheduling is considered more difficult to deal with than scheduling because it involves rapid decision making in response to incidents that occur during operation [45]. Cacchiani *et al.* [7] point out another important difference between scheduling and rescheduling; rescheduling generally has less flexibility because at the time of the disturbance many events have already started and cannot easily be altered. In addition train rescheduling is usually performed on a smaller fragment of the railway network than scheduling [46]. Finally the main reason for not including scheduling research in this chapter is that scheduling is an inherently static process and this thesis is concerned with optimisation in dynamic environments. In rescheduling both the problem itself and/or the objectives may change over time, this is generally not the case in scheduling.

3.3 Railway Rescheduling Literature

One of the main differences in the approaches taken to railway rescheduling, in the literature, is the number of objectives taken into account. For this reason, the literature has been split into two sections, works that consider a single objective and works that consider multiple objectives. In each of these sections the current

techniques used to optimise the objective(s) are presented and discussed. To give an overview of all the previous work both EC and non-EC techniques are considered. Even though this work focuses on EC techniques, other techniques are presented here to put the work into context and to give an overview of the whole area. In this case a technique is classified as EC if it involves a population of candidate solutions that evolve over time, for example GAs employ a population of chromosomes, while ACOs use a population of ants. Techniques such as tabu search (TS), variable neighbourhood search (VNS) or simulated annealing (SA) are not considered to be an evolutionary approach because they are based on a single best-so-far individual (incumbent solution) that is replaced by better solutions.

A glossary of railway terminology and algorithms can be found at the beginning of this thesis on pages xvi and xviii respectively.

3.4 Single Objective Railway Rescheduling Problems

The following literature considers railway rescheduling problems with a single objective. There is no one, agreed upon, objective for this problem and researchers optimise a number of different objectives, for example, minimisation of total delay, minimisation of total weighted delay, minimisation of passenger delay. This range of different objectives is most likely a result of the many different models and scenarios explored in the current research. Different scenarios will have different objectives, either as a result of the scenario being investigated or as a result of input from the companies that sponsor the research. This suggests that there is no one single objective that is suitable for all problems.

3.4.1 EC techniques for solving Railway Rescheduling Problems

The application of EC techniques to the train rescheduling problem is, at the present time, a relatively unexplored research area compared to the application of non-EC techniques. One of the most popular EC techniques to use for railway rescheduling is a genetic algorithm (GA). Research approaches differ in the way the chromosome is encoded. One approach is to use the chromosome to encode a sequence of trains to pass through a junction [47, 48, 2], whereas in other approaches the chromosome encodes a vector of train times [49]. Table 3.1 summarises the different approaches to railway rescheduling using EC techniques.

Table 3.1: Summary of EC approaches for single objective rescheduling problems

Approach	Authors
GA	Ho and Yeung (2000) [47], Ho and Yeung (2001) [48], Semet and Schoenauer (2005) [41], Fan *et al.* (2012) [2], Dündar *et al.* (2013) [50],
DE	Chen *et al.* (2010) [49], Chen *et al.* (2015) [51]
Other EAs	Semet and Schoenauer (2005) [41]
ACO	Fan *et al.* (2012) [2]
ACO Hybrid	Samà *et al.* (2015) [23]

Railway Rescheduling using a GA

Ho and Yeung [47] considered a rescheduling problem on a model of a junction on a metro line. The junction consisted of two train lines converging into one resulting in a decision needing to be made about the order that trains should enter the junction. They used an event-based traffic flow model based on fixed block signalling involving eight trains, four on each line into the junction. They tackled the problem using a GA. Each chromosome represented a feasible sequence of trains to pass through the junction and the starting population was created from the set of all possible feasible solutions. To prevent infeasible solutions from being produced by crossover or mutation, they replaced these two operations with the selection of a nearest neighbour to the best solution. The nearest neighbour replaced the worst solution out of the two population members. They found that their algorithm could find a solution within less than 5% of the optimal but with a much shorter computation time than using dynamic programming (DP). In 2001, Ho and Yeung [48] extended the work by also comparing their algorithm to the performance of simulated annealing (SA) and a tabu search (TS). They found that all methods provided a similar balance between computation time and optimality and all performed better than DP. These studies show the value of using a GA for a single junction scheduling problem, however, the fact that they only consider a simple two track junction with a maximum of four trains on each side of the junction makes it a very limited problem. In addition, the static nature of the problem assumes that no other trains will arrive while the trains are being fed through the junction, which may be an unrealistic situation. Further, using a starting selection pool of all possible feasible solutions, means that the computation complexity and therefore execution time will grow as more trains are added to the problem.

Other researchers have also applied GAs to railway rescheduling. Fan *et al.* [2]

considered a rescheduling problem on the British railway network on the Derby to Birmingham line taking in the North Stafford and Stenson junctions. Their aim was to compare and contrast eight different rescheduling approaches to determine which approach performed best. They compared Brute Force (BF), First Come First Served (FCFS), TS, SA, GA, ACO, DP and decision-tree based elimination (DTBE). A description of these algorithms can be found in the glossary at the beginning of this thesis on page xviii. FCFS is a simple heuristic often used by railway dispatchers [32]. In FCFS, trains are allowed to pass through the junction in the order that they arrive at the junction. In the GA the chromosomes encoded the sequence of trains to pass through the junction. New solutions were produced by using a two-point crossover operator and infeasible solutions were repaired by pushing the train that violated the constraint further back in the train sequence, until the point where the violation no longer occurred. The problem involved 12 trains of 3 types, Class 150, Class 200 and F2-mixed freight trains. Trains had different masses, acceleration, lengths and braking rates. The scenarios differed in the size of the delay and the number of trains delayed. They found that the GA was slightly slower than TS and DTBE, taking an average of 88 s over all the delay scenarios, but gave 28% improvement in total delay penalty compared to FCFS.

Dündar *et al.* [50] also used a GA but this time encoded the chromosomes to represent which out of a pair of trains should access a passing place first on a stretch of single track on the Turkish State Railway. The track was 150km long with 18 passing points and they considered from 6 to 17 trains. Their objective was to minimise total weighted delay, with the weights representing the train priorities. They compared the GA with an artificial neural network (ANN), trained with dispatchers' responses, and an exact model executed on the LINDO software package. Compared to the ANN the GA was able to find the optimal solution for small sized problems in shorter time and to reduce the total delay by around half. The exact method was run on the small problems with up to six trains and the GA was found to produce the same optimal solution. However, in terms of computation time the GA could produce a solution in under 85 s while the exact solution took 320 s. This is an interesting solution to a static railway rescheduling problem, but modelling more than a single track could result in a very complex chromosome.

Railway Rescheduling using Differential Evolution (DE)

Some researchers have applied a DE algorithm to the railway rescheduling problem. A DE is a population based EC technique where individuals in the population consist of a vector of real values. New individuals are generated by adding a weighted

difference between two individuals to a third individual. If the new individual is superior to the comparison individual it replaces it.

Chen *et al.* [49] used a DE algorithm to tackle the problem of rescheduling at a single junction. The junction under consideration, the St. Pancras Midland Road Junction, has two routes into the junction and one conflict point. The aim was to reschedule 24 trains, 10 on one route and 14 on the other route in a 1 hour window. They used a modified DE strategy (DE_JRM), where the modification consists of an additional operation after mutation and crossover that ensures the solutions produced are feasible. The single objective was that of minimising the weighted average delay for passengers and was calculated by considering the number of passengers that alight from the train at the stop after the delay. The test scenarios were generated using a simulator based on the Monte-Carlo method and they tested the performance of the algorithm by comparing it to FCFS.

They found that the DE_JRM performed significantly better than FCFS on both the short and long delay test cases. In the case of short delay scenarios most of the trains could be rescheduled close to the nominal timetable, in the case of long train delay scenarios, the algorithm could significantly reduce the delay by re-sequencing the trains before they arrived at the junction point.

Chen *et al.* [51] also applied the modified DE algorithm (DE_JRM) to a bottleneck junction on the Thameslink. The section of the network under consideration has trains from four different origins converging into the bottlenecks and up to 24 trains per hour in each direction in peak time. The objective was to minimise the weighted average delay, where the weights reflected the priorities of the trains. They compared the results of the DE_JRM with FCFS and with a conventional automatic route setting (ARS) strategy which decides the order of trains to be run based on each train's estimated delay and its weighting. They found that DE_JRM decreased the average weighted delay significantly compared to FCFS and ARS. It could also produce a solution in around 2-3 s.

A limitation of this work is that it assumes all delays are known about in advance and that no new incidents occur during the rescheduling process. In a real life situation, while trains are queued to pass through the junction, later trains may be building up behind them and experiencing knock on delays.

Railway Rescheduling using other Evolutionary Algorithms (EAs)

Semet and Schoenauer [41] tackled the problem of rescheduling at multiple junctions. They modelled the problem as a graph with nodes representing the stations and edges representing tracks between stations. The model used was based on real data

provided by SNCF, the French national railroad company. To solve the rescheduling problem they created a hybrid system using an EA combined with CPLEX. The purpose of the EA is to generate an optimal permutation of trains which, when passed through a scheduler module, produce an optimal schedule. The scheduler module handles the constraints and uses a semi-greedy method to allocate the trains to the schedule, one at a time, in the order given in the EA's chromosome. The scheduler is considered semi-greedy because it only considers optimality at one node (station) level rather than at the global level. The single objective was to minimise the total accumulated delay of the trains. To generate disruption they delayed a train for 10 minutes at a large connecting node around the middle of the simulation period. To test the efficiency of their algorithm, they compared it to a solution obtained using CPLEX alone.

They found that CPLEX took several hours to reach an acceptable solution for an average size problem and that increasing the size or complexity of the problem greatly affected the computational efficiency. In contrast, the EA needed only 15 minutes to reach a good but sub-optimal fitness level. However, they found that the EA was unable to improve the solution after around 15 minutes. They suggest that this is because the scheduler was semi-greedy rather than greedy meaning that train insertions into the schedule were only optimal at the station level rather than globally. They speculate that the system could be improved by allowing it to look ahead to future stations. A limitation of the research is that the system was only applied to one problem scenario, they did not investigate problems with different patterns of delay. In addition, they assumed that all trains had the same priority level, which may not always be the case.

Railway Rescheduling using ACO

Despite the fact that many railway rescheduling problems can be considered to be combinatorial optimisation problems, there seems to have been little work using ACO for railway rescheduling. In their research based on Stenson junction, Fan *et al.* [2] also applied an ACO algorithm to determine the order that the trains should pass through the junction to minimise the delay penalty. They found that the ACO performed slightly better than the GA taking an average of 77 s to find a solution and producing solutions 30% better than those found by FCFS. They concluded that overall ACO gave the closest results to the optimum within a practical computation time.

Railway Rescheduling using Hybrid ACO

Samà *et al.* [23] used a hybrid system to find a solution to the railway rescheduling problem they named the real-time Railway Traffic Management Problem (rtRTMP). They used the ACO algorithm MMAS combined with a local search to discover an effective set of train routes for input to a MILP solver. The rtRTMP involves the detection and resolution of conflicting resource requests after perturbations in the railway system with the aim, in this case, of minimising the sum of secondary delays at the end station.

The objective function used for MMAS was to minimise the sum of overlap between trains requiring the same track resource. Using MMAS to find good candidate solutions for the routes simplified the work performed by the MILP solver. Experiments were carried out in the laboratory using real-world data from the French railway network around the city of Rouen. Each delay scenario had an average of 13 trains and each train had a maximum of 192 routings. The use of MMAS to create the train routes was compared to randomly created train routes on twenty randomly generated delay scenarios with delays of between 5 and 15 minutes. MMAS almost always gave better results than choosing random train routes and it decreased the average objective function value by more than 50 seconds. In 9 out of 20 of the investigated scenarios the improvement was significant.

This work illustrates how selecting good candidate solutions for the trains' routes before resolving the conflicts is a feasible and efficient approach to the problem. However, this is a static problem and considers a one-time delay scenario, it does not take into account other unforeseen delays that may occur during the time period of the problem.

From the above research it is apparent that EC techniques have the potential to make a contribution to the difficult problem of railway rescheduling. However, none of the rescheduling problems investigated makes an allowance for the dynamic nature of the problem. They do not consider the possibility that more trains may arrive or more train delays may occur as trains are running to their rescheduled timetable.

3.4.2 Non-EC techniques for solving Railway Rescheduling Problems

This section considers previous research applying non-EC techniques to the railway rescheduling problem. Table 3.2 gives a summary of non-EC approaches to railway rescheduling. As can be seen by comparing Table 3.2 with Table 3.1, the use of

Table 3.2: Summary of non-EC approaches for single objective rescheduling problems

Approach	Authors
BB	D'Ariano *et al* (2007) [32, 52], D'Ariano and Pranzo (2009) [53], Corman *et al.* (2009)[42, 18], Corman *et al.* (2010) [54], Corman *et al.* (2011)[55, 56], Kecman *et al.* (2013)[19], D'Ariano *et al.* (2014) [57], Samà *et al.* (2016) [33], Rodriguez (2007)[58]
Heuristic	Törnquist (2007) [59], Mazzarello and Ottaviani (2007) [60], Khosravi *et al.* (2012) [61], Espinosa-Aranda *et al.* (2013)[62], Corman *et al.* (2015) [63], Corman *et al.* (2016) [31]
Greedy algorithm	Mascis *et al.* (2008) [35], Törnquist Krasemann (2012) [64]
CPLEX	Törnquist and Persson (2007)[65], Dollevoet *et al.* (2014) [66], Meng and Zhou (2014) [67], Samà *et al.* (2016) [21]
CPLEX Hybrid	Törnquist and Persson (2005) [68], Pellegrini *et al.* (2015) [44], Mladenovic *et al.* (2016) [46]
VNS	Samà *et al.* (2016) [10]
Q-Learning	Šemrov *et al.* (2016) [69]

non-EC techniques is a much more common approach to the railway rescheduling problem than using EC techniques.

One of the most popular non-EC approaches is to use a Branch and Bound (BB) algorithm. A BB algorithm represents the problem space as a tree with branches connecting nodes. A solution to the problem can be found by taking a path from the top of the tree to the bottom node. The algorithm explores branches of the tree and at each branch the algorithm checks the estimated lower bound at that node against the estimated upper bound of the problem. If the node produces a worse solution than the upper bound the algorithm ceases exploration of that branch as it knows that the branch can never produce an optimal solution. In this way the algorithm prunes branches to reduce the size of the search space and to speed up the optimisation process.

Railway Rescheduling using a BB Algorithm

In many of the following works the BB algorithm is paired with an alternative graph of the modelled railway network. The alternative graph was first introduced by Mascis and Pacciarelli [70]. It is used to model decisions about which train, out

of a pair of trains, should have priority access to a shared resource (block section). An alternative graph is defined as $G = (N, F, A)$ where N is a set of nodes, F is a set of fixed arcs and A is a set of alternative arcs. A node is associated with the starting time of each operation. An arc represents which train can enter a block section first. A fixed arc (F) cannot be changed but an alternative arc (A) gives the option to make a choice about which train should take precedence over the other. Each arc has an associated weight, for fixed arcs this may be the running time or dwell time, for alternative arcs the weight represents the minimum headway time between consecutive trains.

Modelling the problem as an alternative graph has the advantage that it can be used to model fairly large networks with several stations for example the Utretcht to Den Bosch railway line on the Dutch railway network [42]. It can also model the track sections in microscopic detail which makes it suitable for the precise operation of trains, however, this makes it difficult to take into account the ongoing impact of the decisions made on the rest of the trains journey. A further disadvantage of the alternative graph is that it assumes the use of GPS sensors on trains to provide real time position data to the dispatching control centre [55]. This may not be available for all trains in the UK at the current time.

It is also unclear how the alternative graph model could cope with the occurrence of more, unforeseen, primary delays. The creation of an alternative graph involves a preprocessing step in which the graph is created using the predefined routes of each train to create the fixed and alternative arcs [62]. After a dynamic change, it is likely that the model would have to be rebuilt with different weightings and with different alternative arc pairings, as an additional primary delay could mean that trains that were in the previous alternative arc pairs may no longer coincide at the same track section while other, newly delayed trains, may now do so.

One of the first researchers to model the train rescheduling problem as an alternative graph and to solve it with a BB algorithm was D'Ariano *et al.* [32]. In fact many other subsequent researchers, such as [42, 18, 54, 56, 55, 19] have used D'Ariano *et al.*'s technique and applied it to different rescheduling problems. D'Ariano *et al.* [32] modelled the block sections of a bottleneck area of the Dutch rail network at Schiphol. The area considered was around 20 km long and included 86 block sections, 16 platforms and 54 trains per hour. The perturbations were simulated by adding delays to varying numbers of trains randomly chosen within the first half hour of the timetable. The delays were all present at the beginning of the problem and no subsequent delays were added over time. They considered the single objective of minimising the maximum secondary delays for all trains at all

visited stations. Conflict was detected by consideration of blocking time stairway graphs. To speed up the performance of the BB algorithm they used dynamic and static implication rules based on the topology of the network. The static implication rules were calculated in a preprocessing step.

It was that their algorithm outperformed both a FCFS and a FLFS (First Leave First Served) heuristic. It reduced the average amount of secondary delay by around 50% in 120 s of execution time. However, the static implication rules required a great deal of problem-dependent knowledge and were applicable to only that particular section of the network. The fact that the topology and rules about the network may change over time, for example due to speed constraints, broken tracks etc., suggests that the implication rules calculated in the pre-processing step, may become outdated while the algorithm is being run. The algorithm could therefore struggle to deal with a dynamic rescheduling problem where more delays occur over time.

D'Ariano *et al.* [52] extended work on Schiphol underground station by the addition of an iterative variable speed module that allows the speed profile of the trains to be adjusted. The aim was to minimise the maximum consecutive delay. Randomly generated perturbations were applied to a varying number of trains with a maximum delay of 300 seconds and an average delay of 67.5 seconds. The BB always outperformed First In First Out (FIFO) and First Out First In (FOFI) and usually outperformed a greedy heuristic. The BB algorithm also resulted in, on average, two less delayed trains than the FIFO heuristic. They found that the variable speed module gave more realistic solutions than maintaining a fixed speed. However, the rescheduling system does depend on the algorithm having exact knowledge of the position of all trains at all times, which may not always be possible, especially in the British railway network. In addition, the work assumed that all delays are known about in advance and that no other delays occur over the period of the perturbation. The complete system takes around three minutes to solve the one hour delay scenario. This is quite fast, however, during that time there could have been significant changes to the trains in the network which are not taken into account in the algorithm.

D'Ariano and Pranzo [53] investigated how to apply their BB approach to a problem with a longer time horizon of nine hours. They compared creating a huge alternative graph covering the whole time horizon (global approach) to decomposing the problem into sub-problems with smaller time horizons (decomposition approach). They found that the global approach gave the better solutions but was only applicable for time horizons of up to three hours. After that time only the decomposition approach could find a solution in a realistic time frame. This is an

interesting approach but begs the question of how valuable it is to attempt to predict so many hours into the future for an environment that is as unpredictable as the railway system. It does not consider that during the optimisation process more primary train delays could occur requiring the algorithm to be run again to find an updated solution. In fact, Törnquist [59] point out that assumptions about traffic behaviour are more uncertain the greater the duration of the planning horizon.

Corman *et al.* [18] modelled the interlocking area of a complex station in the Netherlands, Utrecht Central station, using an alternative graph and solved the rescheduling problem with the BB algorithm of [32]. They found that modelling the track sections through the station as station routes instead of individual block sections gave a more realistic model of the interlocking within the station. This suggests that the alternative graph approach needs fine turning to ensure that it gives the best results for each network area.

The alternative graph and BB algorithm of D'Ariano *et al.* [32] has been implemented within a real-time traffic management system named ROMA (Railway Traffic Optimisation by Means of Alternative graphs). Corman *et al.* [54] investigated improving ROMA by the addition of a TS to determine the train routes before rescheduling the trains on an area of the Dutch Railway network. Their objective was to minimise delay and they considered perturbations in the dispatching area of Utrecht Den Bosch involving 50 km of track, 191 block sections and 21 platforms with up to 40 trains per hour. Randomly selected trains were delayed within the first 30 min of the problem with an average delay of around 320 s. They found that for small delay scenarios the TS could easily find optimal solutions. For larger delay scenarios the TS generated solutions that were more than 15% better than those achieved by the version of ROMA without TS. In addition the TS found solutions in around 20 s whereas ROMA took up to 180 s.

More work has been carried out to evaluate and refine ROMA. Corman *et al.* [55] carried out extensive experiments to demonstrate the effectiveness of ROMA for Utretcht Central station. They compared the results on delay scenarios based on a statistical analysis of the movement of trains at Utretcht station in April 2008. ROMA with the TS outperformed both FCFS and an automatic train rescheduling system (ARI). The ROMA algorithms improved train punctuality by 92% compared to 89% for the FCFS heuristic. In further work, Corman *et al.* [56] considered trains with different classes of priority and found that the algorithm improved the results of the highest priority class at the expense of the lower priority classes.

The alternative graph approach is a microscopic approach which models train movements at the level of detailed train movements and block sections. However,

using microscopic models for busy and complex railway networks can result in long computation times. This makes such models computationally expensive for complex and extensive areas of the railway network. To resolve this problem, Kecman *et al.* [19] developed four different macroscopic models, with reduced levels of detail and operational constraints, to model a large area of the Dutch railway network. A macroscopic model can still capture the movements of trains in a railway network but in less operational detail.

The first model, Model 1, treated all resources as having infinite capacity with no constraints and did not consider headway between train arrivals or conflict between trains using different resources. It assumed that all trains ran at the same speed with fixed running times. Model 2 extended Model 1 by considering arrival headway time and the sequence of arrivals to a timetable point from the same open track segment. Model 3 captured the conflict between trains on different track sections for example at junctions and Model 4 introduced stations where overtaking was allowed to take place. In each case the models were created using an alternative graph, where a decision about which of two trains to run first was modelled by a pair of alternative arcs.

They evaluated the four models against the solutions obtained using the microscopic model created by D'Ariano *et al.* [32]. They considered delay scenarios on the railway corridor between Utrecht and Den Bosch, in the Netherlands involving 200 delay instances based on the Weibull distributions. They found that the size of the alternative graph increased with the number of operational constraints considered in each model. Models 2, 3 and 4 needed twice as many pairs of alternative arcs to model trains run along open tracks compared to Model 1. They also found that the more complex the model the more realistic the delay propagation as the models were able to capture more interactions between trains. Model 4 showed the best performance in terms of feasibility of solutions and gave the solutions closest to the accurate microscopic model. In addition, Model 4 improved the secondary delay, compared to running the trains without any rescheduling. In fact, it almost halved the amount of delay reducing it from 3093 min to 1611 min.

An essential part of ROMA is AGLIBRARY (Alternative Graph LIBRARY), D'Ariano *et al.* [57] applied AGLIBRARY, to real world delay scenarios on the East Coast Main Line. The section of network under investigation ranged from near King's Cross station to Huntingdon station. They investigated 29 different disruption instances of 15 min, 30 min or 60 min time duration and found that their system performed much faster than the commercial solver CPLEX, a best time of 9 seconds compared to 1011.7 seconds, and found an optimal or near optimal solution

in all cases. In contrast, especially on the 60 minute disruption scenarios, CPLEX often failed to find a solution.

A BB algorithm depends upon the efficient estimation of upper and lower bounds for the problem. Samà *et al.* [33] investigated further improvements to AGLIBRARY with the aim of quickly computing a good quality lower bound which could be converted to a feasible upper bound solution. The lower bound solution was found by relaxing the constraints of the model. This involved creating fictitious, shortest path, routes for each train. The upper bound was calculated by replacing each fictitious route in turn with an optimal real route and evaluating the outcome in the alternative graph. They considered delays on two railway networks, the Utrecht-Den Bosch railway network in the Netherlands, involving 50 km of track and 40 trains, and a section of track on the East Coast Mainline in the UK, involving 80 km of track and 90 trains. Their aim was to minimise the maximum consecutive delay, that is to minimise the largest delay at specified locations. They found that, for the Dutch network, they could quickly compute good quality lower and upper bounds to the optimal solutions in a shorter computation time compared to a commercial MILP solver (CPLEX). For the British railway network, which has less alternative train routes for each train, the upper bound value was not always as good as that found by CPLEX, however CPLEX required on average around 15 min to construct the solution whereas AGLIBRARY could find a solution in around 8 s. This is an interesting approach to the problem of finding good lower and upper bounds for the BB algorithm but the creation of fictitious routes and subsequent replacement with real routes seems to introduce another level of complexity to the problem.

The alternative graph combined with a BB algorithm has also been used by Corman *et al.* [42] to examine energy efficient policies at Schipol and Utretcht. The two policies investigated were Wait in the Corridors (WIC) and Green Wave (GW). In WIC the trains are allowed to wait in the corridors between stations and at the station, in the GW trains are only allowed to wait in the stations and are not allowed to continue their journey until the corridor ahead is clear and they can be guaranteed to face only green signals. They simulated a perturbed situation by delaying from 1 to 8 trains that arrived in the first half hour, the delay was a randomly determined value from 50 to 4800 s. The different policies were translated into different alternative graph models. The objective was to minimise the sum of delays and they compared three different approaches to solving the problem; D'Ariano's BB algorithm [32], an automatic train control system (ARI) and FIFO.

BB outperformed ARI on all problems but on the Utrecht-Den Bosch lighter

traffic area there was no difference between the performance of BB and FIFO using the WIC policy. However, BB with the GW policy performed worse than FIFO on this area of the network. This suggests that on low volume traffic areas simple algorithms, such as FIFO, perform adequately. However, on the more congested network at Schipol, BB with the GW policy performed the best suggesting that advanced scheduling algorithms play a part when scheduled traffic becomes closer to the network capacity. GW also performed much better for both Utrecht-Den-Bosch and Schiphol in terms of energy consumption, with 7% and 13% improvement respectively.

Although an alternative graph combined with a BB is a very popular way to solve railway rescheduling problem, other researchers have used a BB with different models. For example, due to the fact that rescheduling trains after a delay is a highly constrained problem, Rodriguez [58] modelled the problem using a constraint programming model. He solved the problem using a truncated BB algorithm where infeasible nodes are pruned. He modelled a junction north of Paris, the Pierrefitte-Gonesse junction and his objective was to minimise the sum of delays. In the scenario investigated, four trains were delayed with a total delay of 1210 s. Compared to the results of decisions applied by the French operator, SNCF, the technique was able to reduce delays by 62-95%. In addition, the solution was found within the time limit of 180s specified by the SNCF as being the maximum acceptable time for an algorithm to produce a result. To model more complex scenarios, they introduced more train conflicts which revealed a limitation of the technique. As the number of trains in the problem increases the number of decision variables and constraints also increases. For example, the simplest scenario with 6 trains has 2870 variables and 2773 constraints, while the most complex scenario with 24 trains has 9801 variables and 10672 constraints. In addition, the number and complexity of the constraints will also increase as the size and complexity of the network increases. These factors raise questions about the scalability of the approach.

Corman *et al.* [6] point out that although BB algorithms appear to be promising for finding optimal solutions on single track or double track lines the branching rule becomes complicated when modelling a railway network.

Railway Rescheduling using Heuristic Methods

Another popular method for solving railway rescheduling problems is by the use of heuristics. Heuristics have the advantage of being able to find solutions faster than an exact method even if that solution is sub-optimal. Heuristics usually involve the implementation of rules for the algorithm to follow. In a time-critical problem like

railway rescheduling, using a heuristic may make it possible to find a solution within an acceptable time-frame.

Törnquist [59] applied a heuristic approach called HOAT to 40 randomly generated delay scenarios on the Norrköping railway network in Sweden. The generated delays were between 5 and 30 minutes. The heuristic works by modifying the order of trains on the track. It allows trains affected by both the primary and secondary delays to 'step-back' and allow up to a maximum of four other trains to use the track section first. In this problem the speed of each train was fixed. They found that in 35 out of the 40 scenario HOAT could generate the optimum solution in less than 15 seconds. However, the remaining solutions struggled to find the optimum within the maximum time frame of 2.5 hours. The problems that HOAT had difficulty in resolving were those where the disturbance occurred in dense traffic areas during peak hours and involved several trains. This suggests that a heuristic that allows a maximum of four trains to use the track section first may not provide enough flexibility to deal with the complexities of the rescheduling problem in dense traffic areas.

As a step towards creating a Train Management System (TMS), Mazzarello and Ottaviani [60] modelled a rescheduling problem as an alternative graph and solved it with a heuristic. In this case, conflicts are resolved by using the AMCC (Avoid Maximum Current C_{max}) rule developed by Mascis and Pacciarelli [70]. Using this rule, the train chosen in an alternative pair is the one that will give the biggest improvement to the length of the longest train path. Once found, a feasible solution is refined by choosing a critical arc (one that results in the largest amount of extra delay) and choosing a different routing option for one of the trains in that arc. They modelled both moving and fixed block track sections within the alternative graph. After resolving the conflict a Speed Profile Generator (SPG) was applied to reduce the energy consumption of the trains by generating the most energy efficient speed profile for the produced scheduling solution. They tested the TMS on a scenario based on the Schiphol bottleneck with 44 km of track and 27 trains in each direction. They introduced entry delays for trains randomly sampled from a 'Pearson T5' distribution. They found that, compared to reference data provided by ProRail, when running 27 trains, travel times were shorter and reliability was higher using the TMS. In addition severe disturbances could be handled more easily and the energy consumption was optimised. When they executed the TMS as a pilot study on the real network they found that a large delay could have been prevented if the driver had not ignored the TMS recommendation of reducing speed to prevent a stop at a red signal. This raises another issue of how to encourage

train drivers and dispatchers to take on board new rescheduling systems, however, although interesting, this complex and difficult question is outside the scope of this thesis.

Khosravi *et al.* [61] considered delays on a congested section of the British railway network near London Bridge, including 15 km of track, five stations and up to 54 trains per hour. They modelled the problem as a MILP based on a disjunctive graph. They solved the problem using a modified shifting bottleneck heuristic (SB) in which each track section is selected in turn and the order of trains on that track section is determined by the heuristic. They considered three ways of selecting the next train to be assigned to a track, selecting the train with the highest delay cost, selecting the train with the earliest entry time to the network or selecting the train with the lowest possible start time. Their objective was to minimise the total weighted delay at the destination where the weights were based on train priorities. Introduced delays ranged from minor disruptions (up to 15 min), general disruptions (15 to 30 min) and major disruptions (over 30 min). They found there were no consistent differences between the three different methods for selecting the next train and that SB outperformed FCFS, in fact FCFS often ended in deadlock. However, SB suffered from long computation times of up to 26 min which may make it unsuitable for real-time delay scenarios. As the results look promising the authors are investigating ways to speed up the algorithm by introducing more efficient heuristics.

Espinosa-Aranda *et al.* [62] considered the problem of rescheduling trains on part of the Spanish railway network, the Renfe Cercanias Madrid, with 93 track sections and an average of 24 trains per hour. They aimed to separately optimise two objectives, the sum of delays accumulated at each station weighted by passenger demand and the makespan (the total time elapsed from the start time of the first train to the finish time of the last train). They modelled the problem as an alternative graph and introduced a heuristic AMDAA (Avoid Most Delayed Alternative Arc) to speed up the process. In this case, for each option in a pair of alternative arcs the option with the lowest passenger delay is selected. They compared the heuristic with the AMCC (Avoid Maximum Current C_{max}), which was also used by Mazzarello and Ottaviani [60]. The delay was simulated by reducing the speed of a randomly selected set of trains by 5% to 30%. They found that AMDAA and AMCC performed similarly and found good suboptimal solutions although they were not as good as CPLEX. AMDAA performed better than FCFS. With CPLEX, the computational cost grew very quickly with the size of the problem. They also found that optimising for delay resulted in poor results for makespan and vice versa, indicating a need for a multi-objective approach when attempting to optimise these two objectives at the

same time.

Corman *et al.* [63] considered passenger travel time as a surrogate for passenger discomfort and modelled delay scenarios on the Utrecht to Den Bosch route with the objective of minimising the total time spent in the system by all passengers. They modelled the problem as an alternative graph with arcs to model connections. Due to the unavailability of real passenger flow data they used realistic average passenger flow data produced by the infrastructure manager. They solved the problem by applying one of three heuristics and comparing it to the results found using CPLEX. The basic idea behind the heuristics is to simplify part of the problem by fixing some variables. All the heuristics decompose the problem into two sub-problems, the train scheduling problem and the passenger routing problem. In heuristic one (H1) the sequencing decisions for each train are fixed offline and only the departure times are optimised. In heuristic two (H2) both the sequencing decision and the start time of each operation are fixed while heuristic three (H3) starts from the solution produced by H2 and then solves the two sub problems by iterating between train scheduling and passenger routing until no further improvement could be found. They found that H3 gave the most promising results. In some cases, CPLEX could not find a feasible solution even after 8 hours of computation.

Corman *et al.* [31] extended the work above by introducing a fourth heuristic H4. Heuristic H4 works in the same way as H3 but has the ability to improve passenger travel time by delaying the departure of connecting trains to prevent passengers from missing their connections. They found that, on a small network, H1 was the slowest heuristic while H2, H3 and H4 were able to deliver a solution within 10 s. Overall, H3 gave the best performance on this network. On a larger network CPLEX alone was unable to find any feasible solutions for the problem due to the large number of binary variables even after 8 hours of computation. In this case H2, H3 and H4 were able to find good solutions even for large and complex instances when the commercial solver failed to find a solution to the problem. However, both H3 and H4 exceeded the maximum allowed computation time of 3 min. They found that the solutions produced reduced travel time at the cost of increasing the number of passenger transfers. Even though this reduced the time the passengers were in the system it may not be ideal in terms of passenger satisfaction as increasing the number of connections may introduce more stress to passengers, especially elderly passengers who may struggle to get off the train and move to the connecting train. A limitation of this work is that minimising total passenger travel time did not prevent some passengers from having very long journeys. These two factors suggest that measuring passenger discomfort just by the travel time may not cover all the

complexities of a passenger's travel experience.

Although the use of heuristics clearly gives an improvement in execution time over more exact methods like CPLEX, none of the above works consider the dynamic nature of the railway network. They do not take into account the fact that a railway network is a constantly changing system and that more delays may occur, and more trains may arrive, while the original delay is in the process of being resolved.

Railway Rescheduling using Greedy Algorithms

Mascis *et al.* [35] also modelled a railway rescheduling problem as an alternative graph but this time solved it with a greedy algorithm. Their aim was to investigate the introduction of a TMS with partially automated traffic control actions. Being partially automated the system could either implement the actions itself or ask for input from a human decision maker. A greedy algorithm attempts to find an optimal solution by simply selecting the best option at each decision point. In this case, the greedy algorithm chooses the train in an alternative arc pair that minimises the increase in delay. If a feasible solution cannot be found the algorithm backtracks to a different alternative arc. The algorithm was tested on a detailed rail simulator on the Breda junction on the Dutch railway network with lines approaching from Rotterdam, Brussels and Breda. In the first test case a train was delayed travelling from Belgium to Breda by between 780 s and 840 s, in the second test case trains coming from Rotterdam were delayed between 800 s and 900 s for standard trains and between 300 s and 360 s for shuttles. Results showed that the greedy algorithm had a greater impact on reducing exit delays compared to FIFO. For test case one, the greedy algorithm reduced exit delays by 47% compared to a 33.4% reduction using FIFO. For test case two the exit delays were reduced by 29.8% using the greedy algorithm and by 23.3% using FIFO.

Törnquist Krasemann in [64] considered the same problem as Törnquist and Persson[65] but this time used a simulation of the network and a greedy algorithm to find the solutions. The justification for using a greedy algorithm was that, in their previous work [65], they found that CPLEX could not always find a solution within a reasonable time. The railway modelled was the Norrköping traffic district with 28 stations, 15 double and 17 single bi-directional track sections containing 48 to 50 trains. The objective was to minimise total final delay. The greedy algorithm performs a depth first search first by quickly constructing a complete branch of a tree to find a good-enough solution. Each node in the tree holds an estimation of the delay for the solution so far. Once the tree is constructed the algorithm uses the remaining allocated time to back-track through the tree to find a better potential

node with a lower cost estimation. The algorithm then branches from this node to attempt to find a better solution. They found that the greedy algorithm could find good-enough solutions very quickly, however, it did not always find the optimal solutions found by CPLEX. However, CPLEX took longer to find a solution and there were occasions where it failed to find any feasible solution.

The above work suggests that greedy algorithms have a role to play in train rescheduling as they can speed up the search process to find a solution within an acceptable time frame. However, their decisions are often crude and unsophisticated, compared to a meta-heuristic like ACO, as they always choose the best option at each decision point and have no mechanism to refine their choices. This means that they cannot guarantee to find the optimal solution.

Unfortunately none of the above work considers the dynamic nature of railway rescheduling and assumes that all delays are known about in advance and that no further disruptions will occur.

Railway Rescheduling using CPLEX

Solving the railway rescheduling problem using the commercial solver CPLEX is a popular approach in the literature. CPLEX often employs the branch and cut (BC) algorithm. A BC algorithm is similar to a BB algorithm but uses cutting planes to improve the optimisation process.

Törnquist and Persson [65] formulated the rescheduling problem as a MILP and solved it using CPLEX. They modelled a section of the South Traffic District of the Swedish railway network which includes 169 stations, 92 freight trains and 466 passenger trains. Because of the lack of information, all stations were assumed to have four platforms. They considered four different strategies for solving the problem which produced solutions with increasing levels of flexibility. Strategy 1 allows trains to swap tracks but maintain the train order. Strategy 2 allows track swaps and allows a change of order if the trains are scheduled to use different tracks. Strategy 3 allows a set number of order changes to take place for trains using the same track sections and strategy 4 uses the full model and places no restriction on the number of order changes for trains using the same track sections. They considered two separate objectives that of minimising total delay and minimising total delay costs for passengers. Delay costs for passenger trains reflected the cost to passengers of the delay and also the potential cost of missing connections. The two objectives were evaluated by running them separately one at a time through the CPLEX program.

They investigated a single disturbance during the morning rush hour and modi-

fied it by changing the magnitude of the delay using different time horizons of 30, 60 and 90 min. They found that even though strategy 4 had more freedom to modify the initial timetable, strategy 3 often found a comparable solution in less computation time. An investigation into the characteristics of the delay scenario found that when the disturbance was less than 30 minutes and where the disturbed train was running into a less dense area strategy 3 performed well compared to strategy 4. However, when the disturbance magnitude was greater than 25 min and when the delayed train was passing into a dense traffic area strategy 4 performed better than strategy 3. This is because the characteristics of this disturbance mean that more trains may become affected, which requires the more extensive modification of the timetable only achievable by strategy 4. The problem with strategy 4 is that it has a much higher computational demand and cannot solve large problems in the time available. They concluded that strategy 3 is the best method as it obtains good solutions in a reasonable amount of time.

The main limitation of this research is the fact that the method used to obtain a solution does not allow the solutions for more than one objective to be produced at the same time. Each objective has to be evaluated separately. In addition the model lacks some detail in that all station segments are assumed to have four tracks and switches between tracks are not modelled. Adding additional complexity with these features may increase the computation time further.

CPLEX has also been used with an alternative graph to solve railway rescheduling problems. Dollevoet *et al.* [66] considered a rescheduling problem at Utretcht station. The objective was to minimise passenger delay. They used the alternative graph to model the microscopic train scheduling problem and an event-activity network to model the macroscopic delay-management problem of deciding which connections to maintain. They iterated between both approaches to find a feasible solution. They considered two types of delay scenarios, small delays of between 1 and 5 min and large delays of between 1 and 15 min. Each train arriving at the station had a 10% probability of being delayed. They found that the two algorithms together could find a feasible solution, however, the iterative method was very computationally expensive. One round of macroscopic delay management followed by microscopic train scheduling took around 6 min for the small delay problem, iterating this several times would result in a very long computation time. For the large delay problem the delay management program had to be limited to 20 min as it took so long to solve it to optimality. It is likely that this iterative approach would take too long to be able to find a solution in a real-world delay scenario.

Meng and Zhou [67] modelled an N-track network as an IP and used CPLEX

and an algorithm based on a Lagrangian relaxation solution framework with a shortest path algorithm to solve several rescheduling problems. A Lagrangian relaxation results in an approximate but simpler problem, in this case the approximate solution was made feasible using priority rules. Their aim was to make a comparison between the outcome of performing rerouting and rescheduling sequentially to the outcome of performing rerouting and rescheduling simultaneously. Their objective was to minimise the total deviation time of all involved trains. Delays were added of between 10 and 20 min. The network considered was a general N-track network with 287.7m of track, 85 nodes (stations, sidings, points etc.), 97 block sections and from 20 to 40 trains. They found that CPLEX could find optimal solutions for smaller cases with 10 or less trains, but when the number of trains was greater than 10 it could not find solutions after 3 hours, thus, revealing CPLEX's difficulty in finding solutions for larger rescheduling problems. In contrast, the Lagrangian relaxation solution algorithm could find solutions for all the problems in around 1.3 min. They found that simultaneously optimising routing and rescheduling improved the upper bound by 10.9%, compared to sequentially optimising the two parts of the problem. Comparing the Lagrangian relaxation solution framework to FIFO gave a 12.10% improvement when there were 20 trains in the problem and 25.9% improvement with 40 trains. The time to find a solution was within 5 min. The model is limited by the fact that they assumed, for simplicity, that the length of a train is always zero and therefore they did not take into account the fact that a train can occupy more than one track section at a time.

Samà *et al.* [21] formulated the rescheduling problem as a series of MILPs based on the alternative graph model. Each MILP was created with a different objective. The objectives included, minimisation of maximum delay, minimisation of total delay, minimisation of the number of delayed train, minimisation of weighted delays, minimisation of the sum of arrival times of all trains at their exit network and minimisation of the travel time of all trains in the network as a measure of energy consumption. They used CPLEX to solve the problem and ran it separately for each objective. Their aim was to produce multi-criteria decision support methodology to allow the dispatcher to have a choice about which solution to implement.The network under consideration was the Dutch railway taking in Utrecht Central station area. It consisted of 300 km of both single and double track,1000 block sections and 200 platforms. In their model, the train speed, train route and dwell time were fixed. They generated sets of entrance perturbations using the Weibull distribution and they considered two scenarios with either 30 or 60 minutes of traffic. The former included 99 trains and the later included 154 trains. The trains were a mixture

of local, intercity and high speed trains. They found that for the 30 min instances CPLEX took on average no more than 15 s to compute an optimal solution, however, for the 60 min instance CPLEX was not always able to compute an optimal solution within 1 hour of computation. This illustrates the influence the size of the problem has on CPLEX's ability to find a solution. The disadvantage of this method is that CPLEX has to be run separately for each objective and although it runs in a very short time, for the 30 minute problem, multiple executions would increase the time taken to find a set of solutions for the dispatcher to choose from. Although their technique produces a number of different solutions for different objectives, the solutions themselves cannot be considered a POS because it is not possible to guarantee that solutions do not dominate each other (see Section 3.5).

Again, none of the above work takes into account the fact that not all delays can be known about in advance and that more delays may occur while the trains rescheduled after the first delay are waiting to pass through the network. If subsequent delays were to occur there is no mechanism in CPLEX to take into account the solutions found in the previous problem to speed up the identification of new solutions. The algorithm would have to be restarted, which is computationally expensive and could result in the loss of potentially useful information.

Railway Rescheduling using CPLEX Hybrid Algorithms

Due to the amount of time CPLEX can take to find an answer, other approaches have combined CPLEX with heuristics or other algorithms in an attempt to speed up the solution generation process.

Törnquist and Persson [68] combined CPLEX with either an SA or a TS in an iterative two level approach to rescheduling trains on a section of the Swedish railway network. The SA or TS was used to determine the order of trains on the track sections. This order was then passed to CPLEX to determine the start and end times for each train on each track section. The area under investigation was that of the Blekinge Kustbana network in Sweden. It consists of single tracked lines with 41 trains in the 30 hour dataset. The movement of the trains on the track sections were modelled as a series of events. To simulate a disturbed scenario they delayed randomly selected trains with intervals of 6-15 min, 16-25 min or 25-35 min. The objective was to minimise the sum of final delays when the trains arrived at their destination stations. They found, as they expected, that the time that the delay takes place has a large effect on the overall delay in the system. That is, the objective function has the largest value for a train that is given a large delay early in the morning, at the beginning of its journey as this train is more likely to meet several

other trains and propagate its delay forwards. They found that TS often performed better than SA, but that both reduced the delay significantly compared to making no modification to the timetable. TS showed an average of 84.38% improvement while SA generated an average of 62.74% for this data set. However, neither of the heuristics reached the optimum found using an MILP model and CPLEX. One of the limitations of the work is that the data set used represents a rather closed homogeneous system. They acknowledge that the performance of their proposed approach could be different in more complex environments. In addition, the use of the linear optimisation model to determine the start and end times of the trains, on each track section, means that as the number of the trains in the system is increased, the number of variables and constraints in the problem would also increase which would have a detrimental effect on the time needed to produce a solution.

Pellegrini *et al.* [44] tackled the rtRTMP on three railway networks in France; the Pierrefitte-Gonesse junction (avg. 14 trains an hour, 174 block section); the Lille-Flandres station, (avg. 31 trains per hour, 734 block section) and the Rouen-Rive-Droite control area (avg. 10 trains per hour, 189 block sections). They modelled the problem as an MILP and solved it using CPLEX with a number of performance boosting heuristics. The heuristics included actions such as initialising the algorithm with an initial solution based on a FCFS greedy method and reducing the number of constraint variables by exploiting the fact that the topology of the railway network imposes precedence relations between consecutive trains. They named their algorithm RECIFE-MILP. The heuristics selected for each problem were determined automatically by an algorithm configuration tool named SMAC. SMAC worked offline to select the most appropriate heuristic by testing the heuristics on randomly selected samples of classes of problem instances. From then on, whenever that class of problem, was encountered the performance boosters determined by SMAC were run on that problem. SMAC was run only once for each railway network to set up the appropriate boosters, and took five days to execute. To test the algorithm they created 30 random scenarios, and randomly selected 20% of trains to suffer a random delay between 5 and 15 min at their entrance to the network. They compared RECIFE-MILP to one of France's current traffic management strategies, that of giving priority to on-time trains and performing no rerouting. Their objective was to minimise total weighted delays at the exit to the railway network. They found RECIFE-MILP was at least 49% better than the current traffic management strategy at Gonesse, 55% better At Lille-Flandres and 70% better at Rouen-Rive-Droite station. The solution was found within the three minute time limit specified by the SNCF, the French operating company.

This work illustrates the advantage of combining CPLEX with heuristics to improve its performance, however, a limitation of the approach is that it may not be possible to identify all classes of problem instances in advance, as other unforeseen problems may occur in the future. This may make it difficult to determine the rules for selecting which heuristic to use in a problem that has never been encountered before.

Mladenovic *et al.* [46] used CPLEX with a constraint programming model to solve a rescheduling problem on a bottleneck section of the Belgrade railway network with both single and double track and mixed passenger and freight trains. To speed up the search process they integrated three types of heuristics into the problem; bound heuristics, separation heuristics and search heuristics. The bound heuristics limit the domains of the decision variables and objective function, the separation heuristics reduce the number of trains in the problem by including only those trains that influence each other and the search heuristics focus on producing optimal partial schedules. All heuristics were run simultaneously. The algorithm was run separately for a number of different objectives including minimising the maximum delay, minimising the total delay, minimising the number of late trains and minimising the total and maximum weighted delay. They compared their algorithm to the results of 147 real traffic scenarios where dispatchers had used the FCFS rule. On average their algorithm reduced the total delay by 24.93%. The work is limited by the fact that they considered the trains journey on only a small area of network. Extending to a larger area of network may have made it infeasible to find a problem in the required time interval of 30 s because of the increase in the number of variables and constraints.

This research shows that hybridising CPLEX with heuristics has the desired effect of reducing the time the algorithm takes to find a solution. However, a limitation of all these works is that even though multiple delays may be considered they are all known about in advance. The research does not address the dynamic and stochastic nature of the railway network. It does not consider that further unforeseen delays can occur over time changing the nature of the problem with each new delay.

Railway Rescheduling using Variable Neighbourhood Search (VNS)

A VNS algorithm attempts to optimise a problem by replacing the best-so-far solution (incumbent solution) with a better solution found by using a local search in the neighbourhood of the current solution. In VNS the neighbourhoods are changed during the search to avoid the algorithm becoming trapped in a local optimum.

Samà *et al.* [10] integrated a VNS within the AGLIBRARY optimisation solver

of the ROMA dispatching support system. They considered four different VNS strategies, VND, General VNS, Basic VNS and Reduced VNS. VND is a deterministic version of VNS where a restricted neighbourhood to the incumbent solutions is explored using a local search and the best solution identified. General VNS is a generalisation of VND where the solutions in the neighbourhood to the incumbent solution are identified and a shaking procedure is performed which involves choosing a solution randomly. In Basic VNS the algorithm first performs a shaking operation where it chooses a random solution from the chosen neighbourhood and then performs a local search in a restricted neighbourhood of that solution. Reduced VNS is similar to Basic VNS but without the local search so that only the shaking operation is performed. Their objective was to minimise the largest positive deviation from the scheduled timetable at relevant locations.

They investigated four different railway networks. An Italian single-track network, a Dutch double-track network between Utrecht and Den Bosch, a busy and complex area around Utrecht Central station, the busiest station in the Netherlands, and a UK double-track railway network on the East Coast Main Line. Twenty traffic disturbances were considered for each railway network with varying initial delays to the trains as they entered the network area.

They found that Basic VNS was the best VNS for these networks and delay scenarios. The inclusion of the local search procedure in this algorithm means that it outperformed Reduced VNS. General VNS sometimes improved the performance of VND because of the inclusion of the shaking procedure which allows it to escape from a local optimum. Basic VNS always outperformed TS, while the CPLEX solver was unable to compute good quality solutions for most of the delay instances. Not surprisingly they found that when the traffic was denser and multiple re-routing options were available the algorithms produced more diverse solutions than when there were fewer trains per hour.

Again, however, they considered only static scenarios. They assume that all delays are known about in advance and that no further trains are delayed during the course of their investigation. As each network is modelled in microscopic details, if the problem changes then the weights on all the arcs in the alternative graph used by ROMA would have to be updated which would be computationally expensive. In addition, they consider only the train movements on the section of track they are currently modelling and do not consider the effect the decisions made in the localised area have on the trains' ongoing journeys.

Railway Rescheduling using Q-learning

Q-learning is a reinforcement learning approach based on a single agent. The agent performs actions which are rewarded or punished and the agent's goal is to maximise its total reward.

Šemrov *et al.* [69] used Q-learning to resolve train rescheduling problems on a 64km section of the Slovenian railway network composed of 23 block sections, 14 stations and 26 trains in three hours. The Q-learning agent's actions consisted of setting the signal aspect to 'stop' or 'go' for each train moving in the block section before the signal. The result of the decision was then communicated back to the agent in the form of a reward or punishment depending on the amount of delay that occurred when the solution was run in a simulator. They considered 50 delay scenarios where random delays (between 5 and 30 min) were added to random trains and 50 extended delay scenarios where the original 50 scenarios were made more complex by the addition of two more delayed trains. All delays were known about in advance. They compared the results to FIFO and to a random-walk where signalling aspects were chosen randomly. Their objective was to minimise total delay. They found that the delay using Q-learning was always smaller than, or equal to, the delay obtained using FIFO and random-walk, in addition, Q-learning always produced a deadlock-free solution as opposed to FIFO and random-walk. However, the Q-learning algorithm took much longer to find a solution, from 3.1 to 15.2 min, depending on the number of algorithm iterations, compared to 0.1 min for FIFO and random-walk. A limitation of the work is that they did not consider the length of the train in the evaluation, they assumed that the length of a train was a single unit so that it could never be in more than one track section at the same time. In a way the Q-learning algorithm is similar to ACO, in that agents find solution and reinforcement is given in terms of how good that solution is, for the ants the reinforcement is encoded in the pheromones, for the Q-learning agent the reinforcement is given directly to the agent. This suggests that using ACO may also have worked in this scenario with the advantage that it may have decreased the execution time because many agents, rather than a single agent, are exploring the search space simultaneously. This work again assumes that all delays are known about in advance and that no further delays occur during the optimisation process.

3.4.3 Dynamic Railway Rescheduling Problems

In all of the above research the rescheduling problem is solved for a set of delays that are all known about in advance. In the real world it is not possible to know

the future and therefore previous unforeseen delays may occur, after resolving the original delay, which will change the problem over time.

The dynamic nature of the railway system is rarely considered in train rescheduling research. D'Ariano *et al.* [52] discussed the fact that speed and location modifications may happen while the algorithm is computing a solution, but concluded that the fast speed of their algorithm means that this is unlikely and therefore that such real-time variation would not have an effect on their rescheduling system. As the BB algorithm used in their work appears to have no inbuilt mechanism to cope with change, using it again after a dynamic change would effectively be a restart of the algorithm and would lose information that could potentially be usefully carried over to the new environment.

Corman *et al.* [55] discussed the dynamic nature of the rescheduling process and the ability of ROMA to deal with it. They suggested that the algorithm should be rerun at time intervals small enough to decrease the error between the simulated traffic and the actual traffic. They also suggested avoiding predictions too far into the future due to the unpredictable nature of the railway network. Unfortunately they do not run any experiments to demonstrate ROMA's ability to cope with dynamic scenarios.

A few works have considered the railway rescheduling problem as a dynamic problem. An early attempt at solving the railway rescheduling problem was made by Ho *et al.* [71]. They considered two types of delay, isolated delays that occur in isolation and consecutive scenarios where new trains arrive at the junction before the algorithm has sequenced and cleared all the previous trains. The second type of scenario could be considered a dynamic scenario because the problem will change with the arrival of the new trains. They used an event-based traffic flow model based on fixed block signalling with ten homogeneous trains. To speed up the processing time they employed three look-up tables, each containing the running times for all possible signalling situations. Their objective was to minimise the sum of weighted delays. They found that their controller performed significantly better than using FCFS. However, in the dynamic delay scenarios they did not attempt to carry the information over from before the change to the new problem and instead regarded the arrival of each new train as a new problem to solve. This may result in the loss of information that could have been useful in the new environment.

Meng and Zhou [5] considered the stochastic nature of railway rescheduling by investigating a problem in which the length of time a single track line is blocked is not known in advance. They used stochastic programming (SP) with recourse to solve the problem. With this technique the scheduling model produced consists of

a schedule for the first period and a set of decision rules (recourse decisions) that define which second period action should be taken in response to different durations of blockages. The first period decision should always result in feasible second period decisions under all possible conditions and should minimise the expected train delay for different random instances of track blockages. The schedules were created with a BB algorithm. They considered a single line track on the China railway network from Laizhou to Shaowu with 138km track, 18 stations and 26 or 53 trains per hour. They found that the algorithm produced solutions on average 18% better than a heuristic based on priority rules and took around 10 min on the low density (26 trains per hour) scenario. However on the 53 trains per hour scenario it took over 15 min to find a solution and the algorithm had to be terminated at that point with a resulting 10% improvement of the SP approach over the heuristic. They also compared the BB algorithm with an expected value-based solution (EV solution) used by real world train dispatchers. In this case dispatchers first make a feasible and suboptimal train meet-pass plan using a predicted value of track blockage and then make adjustments when the time of the blockage is known for certain. They found that their algorithm produced solutions that were 10 to 30 % better than EV solutions.

This is an interesting way to meet the challenge of a dynamic railway rescheduling environment, however making schedules based on all possible outcomes may be too time consuming for application in a real-world dynamic environment with many interacting trains as shown by the fact that the algorithm could not find a solution within the specified time period when there were 54 trains in the problem. Instead of trying to produce all possible solutions it may be more efficient to resolve the scenarios dynamically as they arise thus removing the problem of trying to predict all possible aspects of an unforeseeable future.

The fact that railway rescheduling can be a dynamic problem affects the approach that should be taken to solving it. In the current algorithms used to solve the rescheduling problem there is no provision for retaining potentially useful information from before the change to after the change. In this case, if a further incident occurred it would most likely require a restart of the algorithm which would result in the loss of potentially useful information from the previous environment. Retaining such information could speed up the generation of a good solution in the new environment and could save valuable computation time.

3.4.4 An Alternative Definition of Dynamic Rescheduling

In this thesis a dynamic problem is considered to conform to the definition proposed by Nguyen [8] (see Section 1.2). In this definition a dynamic problem is a problem where changes occur over time and the algorithm has to react to each change to produce a new optimal solution.

In the literature there is another definition of dynamic train rescheduling introduced by Dessouky and Mi [72]. In this case dynamic rescheduling is concerned with the operational scheduling of freight trains in the USA and refers to the process of allocating newly arrived trains to the existing train schedule where information about the arrival of later trains is unknown. In the North American and Australian freight rail industry the schedule of freight trains is not included when the timetable is generated but instead is determined a short time in advance of the departure of the freight train. This is because the exact departure time of the freight train is hard to predict. In Dessouky and Mi's [72] work, the trains currently in the network are dynamically rescheduled without reference to any trains that may arrive later. The problem is solved with a heuristic, they named the Dynamic algorithm, which determines the paths and precedence rules of the existing trains and the newly entered train. They found that their heuristic was able to reduce delay by at least 40% compared to a greedy algorithm. However, the fact that they do not take into account the arrival of later trains means that the solution may only be optimal for that moment in time and any impact the rescheduling solution has on later arriving trains will not be taken into account. In the worse case scenario optimising only the current trains may result in a severely sub-optimal solution for later arriving trains.

Dai *et al.* [73] also considered dynamic rescheduling problems with the same definition of dynamic as [72]. They assume that only a train's information at the moment it enters the network is known and that information about later arriving trains is unknown. This means that the schedule of a rescheduled train is based only on the schedules of trains currently in the network and involves the slotting of new trains into existing train schedules.

They investigated a section of the British railway system on the East Coast Mainline with 28 trains. The problem was simulated in the BRaVE (Birmingham Railway Virtual Environment) simulator, a microscopic simulator written in Java developed at the University of Birmingham. They delayed one train at Alexandra Palace station for 8 mins from 7:27 to 7:35 and resolved the delays by reordering trains at the junctions. Their methodology was based on performance-based supervisory control where two different rescheduling algorithms, FCFS and TS, were applied alternatively to the problem. The idea behind the approach was that using

alternate algorithms may help to find a solution in a broad search space and may help avoid the algorithm becoming stuck in local minima. In addition, it may overcome the problem of a single algorithm being unable to find a satisfactory solution in all possible situations. Once the performance using the current algorithm became unacceptable the alternative algorithm was used. The objective was to minimise total weighted delay, weighted according to train type. They found that using alternating algorithms reduced the total weighted delay compared to running the trains in the order enforced by the timetable (TOE) and to using FCFS and TS on their own.

The use of two alternating algorithms to improve the outcome is interesting. However, this definition of dynamic is different to that used in this thesis. They consider the problem as dynamic because information about later arriving trains is unknown. If delays for those later trains were considered in the problem, then it could be considered a dynamic problem in line with the definition of [8].

3.5 Multi-objective Train Rescheduling Problems

The first section of this literature review considered single objective railway rescheduling problems. Some of this research considers more than one objective, for example Espinosa-Aranda *et al.* [62], Mladenovic *et al.* [46] and Samà *et al.* [21]. However, in this thesis, they are not considered to be truly multi-objective problems as the objectives are resolved separately rather than simultaneously. In this section, previous research into multi-objective railway rescheduling problems is presented and discussed.

In a multi-objective optimisation problem (MOP) with conflicting objectives, there is no single solution that is able to optimise all the objectives simultaneously as an improvement in one objective may result in a deterioration in a conflicting objective. Many researchers have tackled this problem by combining the objectives into a single, often weighted, objective. The purpose of the weights is to indicate the relative importance of each objective to the problem solution. This approach will result in a single solution for each run, however, its disadvantage is that the weights will have to be determined in advance using domain knowledge. In addition, this approach assumes that the relative importance of each objective does not change over time. This may not always be the case. For example, in the early morning rush hour, a train dispatcher may wish to minimise overall delays whereas in the afternoon they may wish to maintain connections for long distance travellers. A more flexible approach is to produce a set of trade-off solutions to provide the decision maker with

a choice of solutions. This will allow them to make a decision as to which solution best matches their requirements at a particular moment in time.

In order to produce a set of trade-off solutions, we need a means of comparing solutions against each other. This is achieved using the concept of dominance [29]. A solution x_1 is said to dominate a solution x_2 (denoted as $x_1 \prec x_2$) if:

1. x_1 is no worse than x_2 in all objectives and

2. x_1 is strictly better than x_2 in at least one objective.

Each solution is compared with every other solution. If a solution is not dominated by any other solution, it is added to the non-dominated set of solutions, also referred to as the Pareto optimal set (POS). The points that the Pareto-optimal solutions map to, in the objective space, is known as the Pareto optimal front (POF) or Pareto Front (PF). The POS is the set of trade-off solutions that are presented to the decision maker. The decision maker can be confident that, in this set, no solution is any better than any other solution in terms of the trade off between objective values and it is only their particular preference at that time that makes one solution better than another.

The investigation into multi-objective problems is important in railway rescheduling as there may be more than one objective that a dispatcher needs to consider when minimising the effects of a disruption. The idea that dispatchers would like to optimise for different objectives was reinforced by Mladenovic *et al.* [46] who found that when dispatchers were given their software to trial, they appreciated its ability to find solutions based on different objectives.

Work by Törnquist [59] found that the objective chosen affects the performance of the algorithm. They discovered that using an objective function of minimising accumulated delays had a tendency to delay more trains compared to an objective function of minimising total final delay. Törnquist Krasemann [74] found that different objectives resulted in different solution structures when they considered the problem of rescheduling on a single-track iron ore line in Sweden. These works show the effect the objective function has on the solutions generated and suggests that optimising for more than one objective may result in a more balanced outcome than considering a single objective that fails to capture the complexities of the problem.

In the railway rescheduling literature, multi-objective problems are less commonly addressed than single objective problems. As with the single objective rescheduling problems, approaches to multi-objective scheduling can be divided into EC and non-EC approaches. Each approach will be considered in turn in the following sections.

Table 3.3: Summary of EC Approaches to Multi-Objective Rescheduling Problems

Authors	Objectives	Approach	Solution
Wegele and Schnieder (2004)[12]	Minimise total delay penalty and gate changes	GA-BB Hybrid	Single
Chang and Chung (2005) [13]	Minimise running time, dwell time, headway, average travel time, timetable deviation, maximise train load	GA	Single
Zhao *et al.* (2015) [14]	Minimise total delay penalty and total energy usage	GA & ACO	Single
Pochet *et al.* (2016) [15]	Minimise deviation from target headway and delay at critical points	GA	Single
Fernández-Rodríguez *et al.* (2015) [17]	Minimise running time and energy consumption.	Dynamic NSGA-II	POS

3.5.1 EC Techniques for Solving Multi-Objective Railway Rescheduling Problems

As with single objective rescheduling problems, EC techniques are less commonly employed than non-EC techniques. Very few produce a POS of trade-off solutions. Table 3.3 gives a summary of current research using EC techniques for multi-objective railway rescheduling problems. In nearly all cases the researchers choose to combine the objectives into a single, often weighted, objective to produce a single solution.

Wegele and Schnieder [12] considered two objectives, minimising total delay penalty and minimising the use of a different gate to that published in the time table. They combined both objectives into a single objective and solved it with a three-part system consisting of a BB method to find the starting solution; a GA to improve the starting solution; and a conflict tree, based on a Unified Modelling Language (UML) sequence diagram, to analyse the trains interactions and to check for conflicts. They evaluated their system on a Petri net model of 1% of the German railways network in Northern Germany which incorporated both single and double track railway lines and included 104 stations with over 1000 passenger and freight trains in a 24 hour period. They found that the algorithm could find a solution very quickly, in around 3 min, and that this solution was only 11.2% worse than an optimal solution created by assuming none of the trains conflicted with each other.

Chang and Chung [13] considered the multiple objectives of minimising train

running time, dwell time, minimum headway, average travel time, timetable deviation and maximising train load. They based their work on a homogeneous Mass Rapid Transit (MRT) system in Taipei. To solve the problem they combined the objectives into a single objective and solved it with a GA where the chromosome encoded the timetable in the form of chains of trains at the switches and signals. To examine how their system could handle rescheduling they considered two scenarios: the first involved a surge in the passenger flow at one station, the second involved delaying the 23rd train in the schedule by 300 s. Their results showed that the algorithm responded to a surge in passengers by increasing the average number of trains that passed by the affected station from 4 to 6. It achieved this by reducing the dispatching headway of the trains and decreasing the running time and dwell time. Their algorithm also responded well to a delay in one of the trains. After 10 minutes, it had produced a new schedule close to the original timetable and reduced the total delay. To make the rescheduling process faster, they employed a chromosome mask which limited the activity of the algorithm to only the trains affected by the disruption. In addition they seeded the rescheduling population with the existing schedule.

This work is limited by the fact that it has been applied only to a fairly simple train scenario, that of homogeneous trains all travelling in the same direction. A further problem with the system is that it relies on knowing exactly where each train is at any particular moment in time. This requires an expensive and reliable infrastructure that allows the precise position of each train to be pinpointed. This is not available in many countries, particularly the UK, at this present time. Further, the stopping condition for the algorithm is that of finding a feasible solution that does not violate running time, headway or dwell time constraints. There is no way of knowing if this is the optimal solution.

Zhao *et al.*'s [14] considered the two objectives of minimising total delay penalty and minimising energy usage. Their aim was to discover an optimal speed profile for each train that followed a delayed train where the speed profile describes the speed that each train should travel between each station. They combined the objectives into a single weighted fitness function with different weights to represent different driving styles, for example weights of 6:4 placed a stronger emphasis on minimising delay and resulted in an objective that gave a priority to reducing journey time whereas weights of 4:6 placed the emphasis on reducing energy usage. They developed a multi-train simulator to test their approach involving four trains along a 27.5 km single track with three stations. They compared the performance of three algorithms, an enhanced brute force (EBF) method, an ACO algorithm and a GA.

The EBF method was enhanced by only considering trains with a speed greater than plus or minus 10 km from the estimated speed to constrain the size of the problem and to reduce the computation time. In the ACO algorithm, each node to be chosen by an ant consists of a train and a speed between two stations. In this way the ants chose speed profiles for each train in turn. For the GA, each chromosome encodes a set of target speeds for the trains. The GA and ACO algorithms were run until a target number of iterations had been reached or there had been no change in the best solution for a set number of iterations. To create the delay scenario, the first train was delayed by 120 s. Only the speed profiles of the trains that followed the delayed train were optimised, the delayed train itself was run as quickly as possible, within the line speed limits, to catch up with the timetable.

They found that both the GA and ACO were able to find solutions close to the optimal found by the EBF algorithm, but with significantly lower computation times. The ACO and GA algorithms took from 7.5 to 10.5 min to find a solution while the EBF took over 29 hours. All algorithms produced solutions better than running without any optimisation, for example the GA produced solutions with a 17.9% reduction in total cost compared to running without any optimisation algorithm. This work is interesting in that it considers both accumulated delay penalty and energy usage. However, using a single weighted fitness function means that to find the results for different driving styles the algorithm had to be run several times with different weightings. In a real-world delay scenario, there may not be enough time to do this.

Pochet *et al.* [15] considered the bi-objective problem of minimising deviation from the target headway and minimising delay at critical points. This allowed them to simultaneously attempt to maintain both regularity and punctuality. Although they had multiple objectives they returned a single solution for each scenario. They were concerned with the problem of managing delay when there are two types of trains in the network; those that use communication based train control (CBTC) and those that do not. In a CBTC system the position of trains can be accurately determined and maintained by controlling the train's speed and acceleration while it is running. Such trains use a moving block signalling system, as opposed to the fixed-block signalling system of non-CBTC trains. This allows CBTC trains to operate with shorter headways and increases the capacity of the railway line. However, mixing the two types of trains together on one line increases the complexity of the rescheduling process. They considered trains travelling in one direction on the East/West line in the region of Paris taking in 14 single track segments, six stations and five trains.

To tackle the problem they developed a GA that encodes the percentage increase or decrease in running or dwell time of each train at each of the stops on its route. The non-CBTC trains were not considered in the chromosome but were incorporated as constraints. They compared the results of the GA with a basic regulation method which consisted of reducing the running time of delayed trains as much as possible until the delay was recovered. They found that the GA improved punctuality and regularity when the objectives were conflicting and when disturbances were large (over 260 s). This is an interesting approach to the problem of dealing with delay scenarios involving a mixture of CBTC and non-CBTC trains. However, it does not change the running times of the non-CBTC trains, which means that there may be a possibility for further improvement if the non-CBTC trains were also allowed some flexibility. In addition, although they used two objectives they returned only a single solution, which implies that the objectives have been combined in some way.

In all of the above work the objectives are combined into a single weighted objective to produce a single solution. Although, in a way, this appears to be an easy and more straightforward approach, it raises questions about how to combine the objectives. The decision about the relative importance of each objective requires domain knowledge and can never be adjusted to reflect a dispatcher's priorities at a particular moment in time. Using a multi-objective algorithm (MOA) instead would enable a set of trade-off solutions to be given to the dispatcher to allow the dispatcher to make an informed decision based on the circumstances at that point in time. In effect a MOA removes the guesswork when setting up the algorithm. This can only be a good thing in terms of giving dispatchers all the information they need to make informed decisions.

3.5.2 Non-EC Techniques for Solving Multi-Objective Railway Rescheduling Problems

Table 3.4 summarises non-EC approaches to multi-objective rescheduling problems. Again, most researchers combine the objectives to produce a single solution. Only Corman *et al.* [16] produced a Pareto optimal set of 'trade-off' solutions.

Albrecht and Oettich [75] considered the two objectives of minimising missed connections and minimising energy consumption. They combined the objectives into a single weighted objective and solved the problem with a DP approach which involves decomposing the problem into a number of states, solving each state to optimality and then combining the solutions into a single solution. The area under consideration was 17 km of track on the Dresden railway line. They considered the rescheduling of only one train in two different scenarios. In the first scenario the

Table 3.4: Summary of Non-EC Approaches to Multi-Objective Rescheduling

Authors	Objectives	Approach	Solution
Albrecht and Oettich (2002)[75]	Minimise missed connections and energy consumption	DP	Single
Walker *et al.* (2005) [76]	Minimise timetable deviation and cost increase from the adjusted crew roster	BB	Single
Flamini and Pacciarelli (2008)[77]	Minimise timetable deviation and maximise train regularity	Greedy	Single
Schachtebeck and Schöbel (2008) [78]	Minimise sum of delays and missed connections	Heuristic	Single
Dollevoet *et al.* (2011) [79]	Minimise sum of delays and missed connections	CPLEX	Single
Acuna-Agost *et al.* (2011)[80]	Minimise delay cost, changes of track/platforms and unplanned stops	CPLEX	Single
Caimi *et al.* (2012) [81]	Maximise number of scheduled trains, minimise delay penalty and broken connections	CPLEX	Single
Corman *et al.* (2012) [16]	Minimise train delays and missed connections	BB-Heuristic Hybrid	**POS**
Veelenturf *et al.* (2015) [82]	Minimise the number of cancelled train minutes and total delay	CPLEX	Single
Toletti et al. (2015)[83]	Minimise total delay at stations, number of cancelled trains, energy consumption	CPLEX	Single
Tamannaei (2016)*et al.* [84]	Minimise the cost of timetable deviation and train cancellation	SA	Single
Umiliacchi et al. (2016) [85]	Minimise energy consumption and delay	Constraint Optimisation	Single
Yin et al. (2016)	Minimise passenger travel time, passenger delay and energy consumption	Approx. DP	Single

feeder train was delayed and therefore the train was able to use the longer travel time to perform longer coasting phases and thereby reduce energy consumption by 23%. In the second scenario arriving trains were delayed by 20 to 60 s at Dresden-Dobritz and at Heidenau resulting in possible missed connections. The algorithm increased the probability of getting the connection at Dresden-Dobritz from 11% to

69% and at Heidenau from 69% to 93%. A limitation of this work is that it is quite a simple scenario as only one train is rescheduled. In addition, in the first scenario, slowing the train down to reduce energy will result in a delay to that train on the rest of its journey, this is not taken into account in the solution evaluation which may give a rather myopic outcome.

Walker *et al.* [76] considered the minimisation of both timetable deviation and crew rescheduling costs. They used a BB algorithm with Column and Constraint Generation to solve a train rescheduling problem after delay on the Wellington Metro Line in New Zealand. They formulated the problem as a linear program with two coupled, but separate, blocks containing variables and constraints for the train timetabling and crew rostering problems respectively and combined the two objectives into a single objective. To create the simulated problem they introduced delays of 5, 15 and 45 min at different times of the day. They found that their solution could reach optimality in 26 s to 110 s and suggest that this time could only decrease as the power of computers increases. They found that the longer the delay the longer the algorithm took to solve the problem, in addition the time of day that the disruption took place influenced the solution time, with disruptions occurring later in the day being quicker to solve. Unfortunately, they did not compare their approach with any other algorithms. They suggest that the length of the time window should be dependent on the amount of disruption in periods of relative calm the system could look a long way into the future, in periods of chaos the system should just resolve for the immediate disruption as soon as possible. However, this is a 'firefighting' approach that may resolve the immediate problem at the expense of future optimality.

Flamini and Pacciarelli [77] considered the bi-objective problem of minimising timetable deviation and maximising train regularity at an Italian metro rail terminus with up to 30 trains. The two objective were solved sequentially, the first step was to execute a fast heuristic to optimise punctuality by routing and sequencing trains through the station. The second step optimised the regularity of train services with the constraint that the result of the first step was unchanged. The first step of the problem was modelled as an alternative graph. Using the graph trains are assigned routes in a greedy fashion so that they reach their destination with the minimum delay and do not interfere with previously scheduled trains. This stage assigns both a route and a set of arrival and departure times to each train. The route and timings are passed to the second stage which adjusts the departure times of the trains to improve the regularity of the train service. The algorithm worked very quickly to produce an answer in around 5 s with 20 trains and found good, suboptimal,

solutions within 4% of the optimal value. However, the authors themselves point out that it would be interesting to attempt to optimise the two objectives simultaneously to obtain a set of Pareto optimal solutions.

Schachtebeck and Schöbel [78] considered the bi-objective problem of minimising the delay and minimising the number of missed connections taking into account the capacity constraints of the tracks. They combined the two objectives into a single objective weighted with the number of passengers and used an integer programming (IP) model to produce a single solution for a case study based on the railway network in the region of Harz, Germany. They broke the problem down into two stages, using a heuristic such as FSFS (First Scheduled First Served) to fix the train order and then solving the second part of the problem of determining which connections to maintain.

This work was extended by Dollevoet *et al.* [79] to also take into account the capacity at the stations but this time for the Randstad railway network in the Netherlands. They used the same combined objectives and again formulated the problem as an IP, this time solving it with CPLEX in an iterative process which oscillates between solving the problem with a given platform assignment and optimising the platform assignment using the timetable and the decisions about which connections to maintain. However, combining the two objectives implies that the weighting for each objective is known in advance and does not change over time. This may not always be the case as the individual objectives may have different precedences at different times of day and/or year.

Acuna-Agost *et al.* [80] modelled the rescheduling problem, on two networks in France and Chile, as a MILP. To reduce CPLEX's execution time they limited the size of the search space by estimating the probability that an event has been affected by the delay. They name the approach SAPI (Statistical Analysis of Propagation of Incidents). Their objectives were to minimise delay cost, changes of track/platforms and unplanned stops. They combined the objectives into a single weighted objective. However, they point out how difficult it is to determine the weights and instead run the algorithm with different combinations of weights to obtain different solutions for different weightings. The French rescheduling problem concerned a section of network passing from Monts to Ruffec with 43 stations and an average of ten trains per hour. They introduced delays of 10, 20 or 30 min to one or two trains at Monts and Ruffec station. The Chilean network consisted of 49 stations and an average of six trains per hour, in this case they introduced delays of 10, 20 or 30 min to one to four trains. The results show that SAPI was viable in practice and could obtain near optimal solutions in a reduced computation time of 300 s or less. The authors

point out that this approach is based on the assumption that all incidents are known in advance and that no further incidents occur, therefore their algorithm would not be able to cope with dynamically changing problems. They suggest that the effect of new incidents could be mitigated by developing an optimisation procedure which could produce solutions that would be robust to further delays. However, in practice this may be difficult to achieve because of the stochastic and unpredictable nature of the railway network and the difficulty in predicting the nature of future delays.

Caimi *et al.* [81] considered the problem of rescheduling trains after a delay at the Berne central railway station in Switzerland. This station has 13 platforms, 6km of track and 1500 trains per day. Their aim was to maximise customer satisfaction and their objectives were to maximise the number of trains assigned a schedule, to minimise the delay penalty and to minimise the number of broken connections. They combined all objectives into a single weighted fitness function. To find alternative solutions they ran the algorithm several times with different weights. To solve the problem they considered rolling time horizons of 20 min to reduce the computational overhead. For each train, they created a set of alternative blocking time stairways based on predetermined routes and precomputed speed profiles. They also created additional stairways by varying the start time of each train in 5 s intervals. They modelled the problem as a binary linear program and used CPLEX to choose one blocking time stairway for each train from its set of alternative stairways to satisfy the constraints and the objectives. They found that their algorithm could find a solution in under 1 min. Unfortunately, as the results were based on a simulation of the environment, they were unable to compare the result to current practice. The fact that blocking time stairways were precomputed may make this solution difficult to apply in a dynamic scenario where, as a result of additional delays, speed profiles and train start times may change over time making the precomputed stairways inappropriate for the new problem.

Toletti *et al.* [83] considered the problem of minimising both energy and delay. They considered a small mathematical model using just four trains and employed a Resource Conflict Graph based on each train's blocking time stairway to model the conflict between trains. Their objectives were to minimise the total delay at stations, minimise the number of cancelled trains and to minimise energy consumption. The objectives were combined into a single weighted fitness function. They found that the rescheduling procedure had a tendency to reduce energy consumption by increasing overall delay, which suggests that the objectives are conflicting and that it may have been better to produce a set of 'trade-off' solutions than a single solution. Although interesting the model used was very small and combining all objectives into a single

weighted objective again raises the question of how to determine the weights.

Veelenturf *et al.* [82] considered the bi-objective problem of minimising the number of cancelled train minutes and minimising the sum of delays on a section of the Dutch railway network where disruptions were a result of blocked sections of track. At the present time the disruptions caused by blocked tracks are solved manually using contingency plans. They combined the two objectives into a single weighted objective, weighting the cancelled trains by a penalty related to the train type or running time and weighting the delay by a penalty for each unit of delayed time. They used a macroscopic model of the network where they omitted detailed signal information. The network had 39 stations, both double and single track and 60 trains per hour. They modelled the problem as an ILP based on an event activity graph where an event represents a train's arrival or departure and an activity represents the links between events. Each activity had an associated minimum duration which modelled the time needed for a resource used by the first event to become free for the second event. They solved the problem using CPLEX. They found that the two objectives were conflicting as the greater the delay in the final solution the less trains had to be cancelled. To find a set of different solutions they had to run the algorithm several times with different amounts of allowed delay. This could have been achieved in one run with a multi-objective EC algorithm. They found that the amount of allowed delay affected the computation time of the algorithm. With only 5 min of allowed delay a solution could be found in less than 2 min, with 10 min allowed delay a solution could take up to 50 min which is obviously infeasible in real-time operations.

Yin *et al.* [86] considered a multi-objective problem on a metro line where their aim was to minimise passenger travel time, passenger delay and the operation costs of trains in terms of energy consumption. The arrival of passengers at each station was modelled as a non-homogeneous Poisson distribution and the total energy usage was modelled as the difference between the tractive energy consumption and the regenerative energy. The objectives were conflicting as the quicker the train travelled a track section the higher the energy consumption and the smaller the delay. The decision variables were the arrival and departure time of the affected trains at each station and their aim was to generate a set of arrival and departure times for the affected trains that could maintain a safe headway between trains and could minimise all the objectives. To solve the problem, they used an algorithm based on an approximate dynamic programming (ADP) technique and weighted each of the objectives to obtain a single solution to the problem. They found that their algorithm outperformed a heuristic (HEM) based on a practice used by the metro

dispatcher in a perturbed situation, where the arrival times and departure times of all trains affected by the delay are postponed. On a small numerical example, the algorithm performed similarly to a CPLEX implementation but in half the time. They implemented the algorithm on a model of the Beijing Metro Yizhuang line and found that again it outperformed HEM in terms of passenger travel time and delay but with a slightly higher energy consumption.

A drawback of the work is that the algorithm produces a single solution whereas they really wanted a set of solutions that reflect different weightings for each of the objectives. To find this set of solutions they were forced to run the algorithm again with different weights for all of the objectives. Each new run of the algorithm would increase the time taken to find a solution. In addition, there is no restriction on the solution that prevents trains from departing earlier than their scheduled arrival time, in a metro model where passengers turn up and catch the next available train this is less important than in a railway network where trains departing earlier than their scheduled departure could cause huge inconvenience to train crew and passengers.

Tamannaei *et al.* [84] considered the bi-objective rescheduling problem of minimising the cost of timetable deviation and train cancellations. They combined the objectives into a single objective to give a single output. They considered the problem of rescheduling trains on two double track railways in the Iranian railway network, after one of the block sections used by the trains had become blocked and impassable due to an incident. Their approach is unusual in that they imposed a cut-off time after the incident (named the affecting threshold (AT)) after which none of the trains were allowed to be rescheduled. The purpose of setting this threshold is to restrict the incident effects to a predetermined time threshold which guaranteed the punctual revival of the original timetable. The drawback of doing this is that it limits the time available to reschedule trains and results in larger delays to some trains. Trains were cancelled if their delay was so great that they had not yet departed from their origin at the time the incident occurred. The trains whose schedules were not allowed to be changed acted as constraints to the trains that could be rescheduled. They converted the problem to a MILP model and solved it with SA. To find new solutions in the neighbourhood of the incumbent solution, two trains were randomly selected and their departure orders were switched. The departure and arrival of all the other trains at all the block sections was then replanned to accommodate the change. Experiments were carried out on a model of the Bafgh-Sirjan and Tehran-Mashhad railways. In each case they blocked a section of the track for a fixed amount of time, for example the first incident in the Bafgh-Sirjan railway occurred in block section 4 from 9:10 to 10:50 am. They found that

the size of the AT influenced the costs. The closer the AT was to the end time of the incident, the faster the recovery of the original timetable and the greater the costs of recovery and vice versa. They also found that SA could achieve good solutions for different instances in short times while for some large-scale problems CPLEX was unable to find even one feasible solution after half an hour. Again this solution suffers from the limitation of combining both objectives into a single objective.

Umiliacchi *et al.* [85] considered the problem of minimising energy consumption and delay for two trains on a single line of track. On a single-track line trains can only pass each other at a passing loop, a delay to one train means that they will not coincide at the passing loop at the correct time for one train to overtake the other. In this case, one of the trains will have to stop at a red signal and wait for the other train to overtake it at the passing loop. Accelerating this train back to speed again requires a significant amount of energy. The objective of the work was to minimise energy consumption and delay. In this work, delay was treated as a constraint thus converting the multi-objective problem to a single-objective problem to produce a single solution. The railway line in the study was 79.4 km of single track line between Aberdeen and Inverness. The two trains considered were a class 158 passenger train and a class 37 freight train. For the delay scenario, the passenger train was delayed by 200 s. To solve the problem both a macroscopic and a microscopic simulator were employed. The macroscopic simulator found a set of feasible speed profiles for the freight train, the speed profile with the lowest energy consumption was then fed into the microscopic simulator to model the movement of both trains on the track. This made it possible to calculate delay and to ensure that the solution met the constraints. If the speed profile did not satisfy the constraints, the next speed profile in the set was chosen and run through the microscopic simulator. They found that, compared to the non-optimised solution, the optimised solution resulted in a 9% reduction in combined energy consumption with delay removed for both trains. In fact both trains now arrived fractionally early, the passenger train 1 s early and the freight train 14 s early compared to 13 s late and 1 s early for the non-optimised solution. This is interesting work and an inspiring way to use a macro-simulator to find a solution that can be validated by running in the microscopic simulator, as it reduces the number of runs required by the microscopic simulator and thus reduces computation time. However, the problem is limited in that only one train is optimised. In a real-world delay scenario often several trains may have to be rescheduled to resolve the perturbation.

All of the above approaches result in a single output instead of a set of Pareto optimal set of 'trade-off' solutions. Producing a POS would automatically give

solutions with different emphasis on each of the objectives and would guarantee that no one solution is better than any other in all of the objectives. This would empower the dispatcher to be able to make an informed decision as to which solution to choose depending on their priorities at the current moment in time. In addition, combining solutions using weights raises issues about how to determine the weights. Some researchers tackle this by running the algorithm several times with different weights, for example [80, 81, 82, 86]. However, rerunning the algorithm will increase the time taken to find solutions which may make the process too computationally intensive to be used in real-time decision support software system.

3.5.3 Multi-Objective Railway Rescheduling that Produces a Set of Pareto Optimal Solutions (POS)

Corman *et al.* [16] are among the few to produce a POS of 'trade-off' solutions as the output of their algorithm. They considered a bi-objective problem on a section of the Dutch railway taking in the 20 platform station of Utrecht and including 10 km of track of each of the five main lines leaving the station. The scenario involved 79 trains and 451 block sections or platforms. The two objectives they considered were that of minimising train delays and maximising the number of retained passenger and rolling stock connections. The two objectives are conflicting, if a connected train is delayed, its connecting train will also have to be delayed to maintain the connection, therefore the more connections that are retained the greater the secondary delays to the trains in the problem. They model the problem as an alternative graph with the addition of connection arcs wieghted with the minimum time separation between two connected events.

They first solve the single objective problem of minimising delays using the D'Ariano's branch and bound algorithm [32] and then use two heuristic algorithms, that act similar to a local search, to refine the solution in order to find the POS. They experimented with two different algorithms; Add and Remove. Both algorithms maintain an archive of non-dominated solutions (Z) which is returned at the end of the search. The first step of running both algorithms is to insert a starting solution into Z. This solution is used as a base for the search to find neighbouring solutions by iteratively adding a single connection constraint in the Add algorithm and removing a single connection constraint in the Remove algorithm. Any solutions that dominate the solutions in Z are added to Z and any dominated solutions are removed. Add differs slightly from Remove in that if two or more neighbours yield the same maximum consecutive delay an additional neighbour is generated. This modification was found not to improve the solutions generated by Remove and so

was omitted from the Remove algorithm.

The 25 perturbation scenarios used in the investigation were generated using the Weibull distributions. Each scenario was run with one of three connection scenarios. The first connection scenario involved 12 passenger transfer connections and 7 non-relaxable stock circulation connections; the second scenario involved 12 passenger connections and 12 relaxable rolling stock connections and the third scenario had the same connections as the second scenario but with larger weights on the connections. Larger weights increase the time separation between two connections and results in a more serious propagation of consecutive delays. They found that both algorithms performed well. On scenario 1 both Add and Remove performed as well as an exhaustive search in less than 1% of the computation time. They found that Add was more efficient than Remove and outperformed Remove on the third scenario. The results are interesting and show the value of using multi-objective algorithms for making complex, real-time rescheduling decisions. However, representing the problem as an alternative graph requires that all the track sections rather than just the track sections that are involved in the decision making are modelled. This results in a very large graph, 1847 nodes, 2156 fixed arcs and 4773 pairs of alternative arcs in this problem. Extending the graph to cover a larger area of the network would involve many more nodes and arcs with a corresponding increase in computational complexity. On the network under investigation the Add algorithm takes an average of 18.71 min to execute while the Remove algorithm takes 31.12 min on the most complex scenario. Extending the alternative graph to cover a larger area of the network would increase the execution time to a level that would most likely be unacceptable in a real world rescheduling situation.

A further limitation of this work is that the problem is static, it assumes that the delays occur at the beginning of the problem and that no more delays occur over the duration of the scenario. In a real-world scenario, there may be further primary train delays which will change the nature of the problem under investigation.

3.5.4 Dynamic Multi-objective Train Rescheduling

None of the above multi-objective research considers the dynamic nature of the train rescheduling problem. Research carried out by Fernández-Rodríguez *et al.* [17] is unusual in that it produces a POS for a multi-objective railway rescheduling problem and also considers the dynamic nature of the problem. They addressed the problem of optimising the speed profile of a single train on a 85.4 km long section of track on a Spanish high speed line. The problem changes over time as the distance to the end of the journey decreases. In this work the delay was caused by a reduced

speed limit for a 14 km long section of track. To address the problem they employed a dynamic form of NSGA-II (DNSGA-II) so that random solutions from before the change were used as a starting point to the algorithm after the change. They used two different versions of DNSGA-II. In the first version (DNSGA-II-A) the random solutions were used directly to replace the population, in the second version (DNSGA-II-B) the random solutions were mutated before being used to replace the population. The algorithms produced a POS of speed profiles for the train at the current moment in time. The solution implemented was the solution with the lowest energy consumption that resulted in the smallest delay.

They compared the results of running DNSGA-II with NSGA-II using hypervolume, which measures how much of the objective space is dominated by the members of the POS [87]. Both versions of DNSGA-II outperformed NSGA-II showing the benefit of carrying over information from before the change to after the change to improve the quality of the POS obtained. DNSGA-II-A, where the random solutions were not mutated, performed better than DNSGA-II-B. Using DNSGA-II-A the total energy consumption was reduced by 5.4%. However, this work is limited in that it considers only one train and assumes that the train does not interact with any other trains on its journey. Executing the algorithm may become very costly and complicated with several trains in a delay scenario especially if the trains came into conflict with each other. It would have been interesting to have extended the dynamic aspect of the problem by introducing more delays over time.

From the above literature, on single and multi-objective railway rescheduling, two things are apparent:

- The dynamic nature of the railway rescheduling problem is very rarely taken into account in previous literature and, if dynamic problems are presented, for example [71, 5], there is often no attempt to carry over information from before the change to help generate solutions after the change. This illustrates a gap in the literature. Fernández-Rodríguez *et al.* [17] demonstrated that retaining information between changes is beneficial to the performance of the algorithm. This suggests that there is a need for more research into the application of algorithms that can retain information between changes, to dynamic railway rescheduling problems.

- Most multi-objective railway rescheduling combines the objectives into a single objective to produce a single solution rather than a set of 'trade-off' solutions. In addition, very few researchers consider dynamic multi-objective rescheduling problems, apart from Fernández-Rodríguez *et al.* [17] and they focused on

only one train. As railway rescheduling problems may involve more than one objective and are, due to the unpredictable nature of the railway system, often dynamic, there is a need for more investigation into dynamic multi-objective train rescheduling problems involving multiple trains.

An algorithm that is capable of addressing both dynamic and multi-objective rescheduling problems is ACO. To deal with the dynamic nature of the rescheduling problem, pheromones can be retained from before a change to help guide the search in the next dynamic change period. In this way, useful information can be retained between changes. To deal with multi-objective rescheduling problems, ACO can be easily modified to address each objective by the addition of multiple pheromone and/or heuristic matrices or by using several colonies of ants. In the following sections research using ACO for dynamic and multi-objective problems will be presented to illustrate the potential of ACO for solving such problems in the railway industry.

3.6 ACO for Dynamic Rescheduling

ACO has not previously been applied to dynamic railway rescheduling problems. However, there is a similarity between the combinatorial optimisation problem of resequencing trains through a junction or a station and the Travelling Salesman Problem (TSP). In both cases the aim is to find a sequencing order for a set of entities. The TSP can be made dynamic by changing the number of cities [24, 26], or by changing the distances between cities [27] over time. ACO has been successfully applied to the dynamic TSP (DTSP), which suggests that it may also be suitable for dynamic railway rescheduling problems.

Guntsch *et al.* [24] investigated the ability of pheromone modification to handle a DTSP. To make the TSP dynamic, they inserted or deleted a city after 250 or 500 iterations. They found that the ACO algorithm could successfully handle the dynamics of the problem and that modifying the pheromones in the area of the change improved the solutions when the changes occurred at a high frequency.

Eyckelhof and Snoek [25] investigated a dynamic version of the TSP where changes were introduced over time by simulating traffic jams on some of the routes to the cities, this had the effect of increasing travel time to those cities. They found that 'smoothing' the pheromone values, by reducing high values and increasing low values, in the area close to the change, led to good results when the traffic jams occurred with high frequency. However, they also found that often the ACO algorithm without any modifications was able to solve the problem when the number of

cities in the problem was small.

Mavrovouniotis and Yang [26] applied ACO to the DTSP, where the dynamic environment was generated by removing half of the cities from the problem and replacing existing cities with the removed cities. The number of cities in the problem did not change overall as the number of cities removed was always the same as the number of cities replaced. They found that an ACO algorithm, modified with a local search scheme, performed well on this problem.

Mavrovouniotis and Yang [27] also investigated a version of the DTSP with simulated traffic jams. To maintain diversity after a change they transferred knowledge from previous environments to the pheromone trail by the use of immigrant ants. The tours of the immigrant ants were included in the pheromone update after a change which meant that knowledge from before the change could be carried forward to the next change. They found that algorithms with immigrants worked better than those without on a randomly changing DTSP. Mavrovouniotis and Yang [28] also found that using multiple colonies of ants with their own pheromone table significantly improved the performance of ACO on most of the DTSPs they investigated.

The above research suggests that ACO has a role to play in dynamically changing environments and that it may be usefully applied to real-world dynamic problems such as railway rescheduling.

3.7 ACO for Multi-objective Problems

The modification of ACO algorithms to make them suitable for multi-objective problems is a popular area for research. Although ACO algorithms were originally designed for single objective problems, their flexibility in allowing multiple colonies, multiple pheromone matrices and multiple heuristic matrices makes them very suitable for problems with more than one objective. In addition, as ACO is population based approach, the set of trade-off solutions can be found in a single run of the algorithm.

Current work with multi-objective ACO (MOACO) algorithms concentrates mainly on benchmark problems such as the bi-objective TSP (bTSP) [88, 89, 90, 91, 92], the multi-objective knapsack problem (MOKP) [93, 92], multi-objective Job Shop Scheduling [94, 95, 96], production and maintenance scheduling [97, 98] and the vehicle routing problem with time windows [99]. Approaches vary in their use of multiple ant colonies [95, 98, 92], multiple pheromone and heuristic matrices [94, 88, 89, 90] or a combination of both [91].

Results have been impressive and many MOACOs have outperformed some of the most popular multi-objective genetic algorithms such as NSGA-II and the Strength Pareto Evolutionary Algorithm (SPEA2). For example, Berrichi *et al.* [98] found that their MOACO algorithm applied to a joint production and maintenance scheduling problem produced better solutions than SPEA2 and NSGA-II in terms of spread and convergence and was able to generate solutions in the two extents of the POF. They suggest that is because of the fact that the ACO algorithm, in contrast to NSGA-II and SPEA2, can make use of heuristic information.

Investigations using real-world problems are less common but have had some interesting applications. One of the earliest of these was by Mariano and Morales [100], who adapted the ant-Q algorithm to create an algorithm capable of solving the bi-objective problem of designing water irrigation networks. The two objectives in this case were to maximise crop profits, and to minimise the cost of the irrigation network. They found that their algorithm could produce solutions better than those found by optimising each objective separately using distributed reinforcement learning. Doerner *et al.* [101] applied a multi-objective version of Ant Colony System (ACS) to a real-world portfolio selection problem. This problem involves selecting investment projects to make up an investment portfolio. In this case, they used a single colony and a pheromone matrix for each objective. Their algorithm outperformed both Pareto Simulated Annealing (PSA) and NSGA on a real-world problem involving 30 projects.

MOACO algorithms have also been applied to the real-world problems of multi-objective multicast routing [102], generating flight trajectories in hazardous weather conditions [103], task scheduling for grid over optical burst switching (GOBS) networks [104], time and space assembly line balancing at the Nissan plant in Spain [105] and mapping virtual machines to physical machines in a cloud computing environment [106]. However, they have not as yet been applied to a multi-objective problem in the railway industry.

3.8 ACO for Dynamic Multi-objective Optimisation Problems (DMOPs)

There has been very little investigation, so far, into using ACO for DMOPs. Colson *et al.* [107] are among the minority when they addressed the problem of micro-grid power management. They considered two objectives, environmental emissions and the cost of power generation, in a dynamic environment that varied in terms of changing power demands and power availability. Using a simulator that could

create 30 second sampling instances they varied the power demands and the local conditions such as wind speed and solar radiation energy to make the problem a dynamic one that varied over time. Between sampling instances they retained the previous pheromones and relied on evaporation to remove old decisions. To cope with the multi-objective nature of the problem they employed multiple colonies of ants, from 5 to 20, with 50 ants per colony. The work was a proof of concept, their aim was to show that ACO could be used for the micro-grid power management problem, therefore they do not compare their algorithm with any other algorithms. Unfortunately, they also did not consider how changing the characteristics of the dynamics of the problem affected the performance of the algorithm.

3.9 Summary

From an analysis of the recent railway rescheduling literature, it is possible to make a number of observations. The first is that EC techniques are less commonly used than non-EC techniques. This may be because EC techniques cannot be guaranteed to always produce the optimal solution. However, EC techniques have an advantage that they can be stopped at any point to return a 'good-enough' solution even if it is not an optimal solution. Corman *et al.* [6], in their review of algorithms for railway traffic management, point out that finding close-to-optimal solutions in a realistic time frame may be enough to meet the requirements of dispatchers attempting to resolve delays.

The second observation is that approaches based on a mathematical model and solved with CPLEX often fail to find a solution in a reasonable time frame in complex delay scenarios, sometimes taking up to several hours [54, 64, 84]. The approach often has to be modified by combining CPLEX with a heuristic [54, 44, 23] which means that the optimal solution cannot be guaranteed.

Thirdly, it is apparent that the dynamic nature of the railway rescheduling problem is very rarely taken into account in railway rescheduling research and if dynamic problems are presented, for example [71, 5], there is no attempt to carry over information from before a change to help generate solutions after the change.

The fourth observation is that most multi-objective railway rescheduling research combines multiple objectives into a single objective to produce a single solution rather than a set of 'trade-off' solutions. This may not satisfy the needs of the railway dispatcher who may prefer to observe the effect of different objectives on the rescheduling solutions before making a decision [46].

Finally, there has been very little work on using ACO for railway rescheduling

problems despite the applicability of ACO to combinatorial optimisation problems [22]. Researchers that have applied it [2, 23] found that it worked well. There is a need to extend this work to explore the application of ACO to both single and multi-objective railway rescheduling in dynamic environments. The fact that ACO algorithms have previously shown good results for both dynamic and multi-objective problems suggests that they may also be applicable to problems that possess both dynamic and multi-objective characteristics.

In the next chapter, the application of ACO to a dynamic junction rescheduling problem is described and experiments are presented that suggest that ACO may have a valuable role to play in railway rescheduling in dynamic environments.

Chapter 4

Railway Junction Rescheduling in Dynamic Environments

Train rescheduling after a perturbation is a challenging task and is an important concern of the railway industry as delayed trains can lead to large fines, disgruntled customers and loss of revenue. Sometimes not just one delay but several unrelated delays can occur in a short space of time which makes the problem even more challenging. In addition, the problem may be a dynamic one for, as trains are waiting to be rescheduled at the junction, more timetabled trains will be arriving, These trains may have different priorities to those already waiting to be rescheduled, which makes the problem a dynamic one that changes over time. The rescheduling of trains after a perturbation is usually dealt with by human controllers [2], who often use simple rules such as FCFS [32]. Although FCFS may resolve the immediate problem, it may not be the optimal solution in terms of minimizing the effect of a train delay in a dynamically changing environment.

The work in this chapter covers two separate investigations. The first uses the original Stenson Junction Train simulator where only the trains in the junction are considered. In this case only the ACO algorithm P-ACO is implemented and the results are compared with FCFS. The second considers an extension to the problem where not only the trains in the junctions but also the trains at the feeder stations to the junction are taken into account. In addition, in the extended version, multiple delays are introduced in each change period and several different ACO algorithms are applied to the problem. In the extended problem the algorithms not only resequence the trains at the junction but also resequence the trains at the stations, which is considered to be a first step towards expanding the problem to consider a larger area of the railway network.

This railway rescheduling problem is referred to as the dynamic railway junction

rescheduling problem (DRJRP). In the DRJRP, the environmental change is a result of the arrival of new timetabled trains while the original trains are waiting to be rescheduled at the junction.

The unique contributions of the work in this chapter are;

- The creation of a benchmark problem and simulator to investigate dynamic railway rescheduling problems.
- A contribution to the field of understanding of how ACO algorithms can be applied to dynamic railway rescheduling problems.
- A contribution to the field of understanding of the effect the characteristics of the delay, in terms of magnitude and frequency, have on the ability of the algorithms to solve dynamic rescheduling problems.

In addition, it is shown in this work that in situations where it is impossible to repair the ant solutions after a dynamic change, elite immigrants can be used to carry information from before the change to the next change period, with no detriment to the algorithm's performance.

This work was previously published in [1] and [38]

4.1 Description of the Problems and Simulators

In the following sections the problems under investigation, the DRJRP and the Extended DRJRP, are described in detail, followed by a description of the simulator used to model and investigate each problem.

4.1.1 The Dynamic Railway Junction Rescheduling Problem

The DRJRP is based on a static benchmark scenario created by Fan *et al.* [2]. It is concerned with a section of track on the Derby to Birmingham line which takes in the North Stafford and Stenson Junctions. Both the junctions are 'flat junctions' in that the merging railroad tracks require that other trains cross over in front of opposing trains on the same level. Two trains can pass through the junction at the same time as long as this does not cause conflict with any other trains. In this work, the benchmark scenario has been extended to make it a dynamic rescheduling problem by introducing more timetabled trains while the original trains are waiting to be rescheduled at the junction. The perturbation to the system is based on the

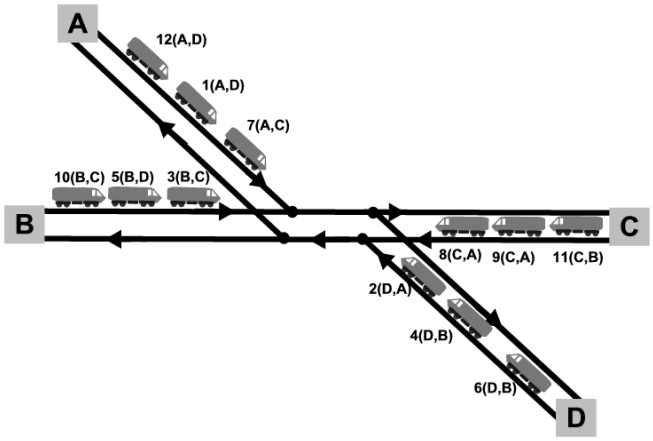


Figure 4.1: The junction before a dynamic change.

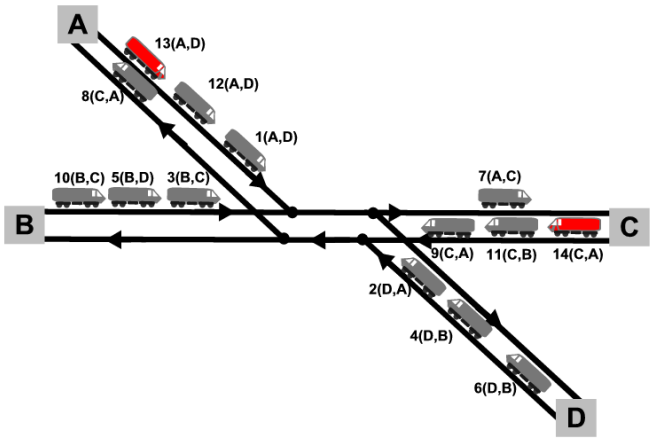


Figure 4.2: The junction after a dynamic change.

second of Fan *et al.*'s delay scenarios [2]. In this scenario, the disruption is caused by train 1 being delayed by 5 minutes, which means train 7 arrives before it on track A. Fig. 4.1 shows a diagram of the static junction with the perturbation.

Fig. 4.2 shows the junction after a dynamic change. Trains 7 and 8 have passed through the junction, but more timetabled trains have arrived while the remainder of the trains are waiting to be rescheduled. Train 13 has arrived on route A while train 14 has arrived on route C. The problem has changed as there is now a different combination of trains to sequence through the junction.

Each train has a delay penalty associated with it, which is the cost in pounds sterling per minute that a train company has to pay if the train is delayed. This is the same objective used by Fan *et al.* [2]. The aim is to find the best order of trains

Table 4.1: Train Arrivals at Station D

Time Passed	Platform 1	Platform 2
10 minutes	Train 14	Train 16
	Train 18	
20 minutes	Train 26	Train 28
30 minutes	Train 30	
40 minutes	Train 38	Train 40
	Train 42	
50 minutes	Train 50	Train 52
60 minutes	Train 54	

to pass through the junction that minimises the overall cost of the delay.

4.1.2 The Extended DRJRP

In the second investigation, the original DRJRP was extended to introduce more delays to the simulator over time. In the extended version, trains added at the station during a dynamic change may also be delayed. This means that instead of dealing with just one delay at the beginning of the simulation, the algorithm also has to deal with additional delays that occur over time. Introducing these delays relies on there being enough trains arriving at the station to ensure that a train arriving after its scheduled time slot at a station has an impact on the problem. For this reason, only the high magnitude dynamic problem, where 8 trains are introduced at each dynamic change, is considered in this extended DRJRP problem. It is only when several trains are arriving at a station within the same time frame that a station delay can be engineered. In fact an investigation into the pattern of arrival of the trains at each change showed that it is only at Station D that sufficient trains are present at the station within the same time-frame to allow the introduction of station delays.

Table 4.1 shows the pattern of trains that arrive at Station D when 8 new trains are arriving every 10 minutes. After 10 minutes have passed (dynamic change 1), three trains arrive at station D: trains 14 and 18 arrive at platform 1 and train 16 arrives at platform 2. The first delay at this station is simulated by switching the arrival order of the two trains on the same platform; trains 18 and 14. A further delay can be introduced by also switching the arrival order of the two trains arriving after 40 minutes has passed (dynamic change 4). This involves switching trains 42 and 38. Delays after the fourth change are not introduced because they would not

be visible in the low frequency dynamic scenarios where only four dynamic changes take place over the period of dynamic change.

4.1.3 The Stenson Junction Train Simulator

Any optimisation algorithm requires a means to test the effectiveness of its solutions in terms of the problem objective. In order to achieve this, a train simulator has been developed, which allows the trains in each train sequencing solution to be run through the simulator to obtain the total delay penalty for that order of trains.

As in the original benchmark scenario used in [2], it is assumed that the junction is clear at the start of the simulation and that each train begins at set distance from the junction. However, in contrast to Fan *et al.*'s simulator, which used a moving block technology, this simulator uses an automatic fixed block technology. The moving block technology assumes a safe distance is calculated around each moving train by an area computer which knows the location of all other trains in the area, whereas the automatic fixed block technology works by preventing trains from entering track sections already occupied by other trains. Modelling the junctions using the fixed automatic block technology is an attempt to make the simulation of the junctions as realistic as possible; the moving block technology has limited implementation in the British railway network although it is in use on a few lines, such as the Docklands Light Railway in London.

The resolution of conflict at each junction is modelled by a simulated interlocking system. This prevents a train from entering the junction unless it is safe to do so and is necessary because trains on some routes cross the path of trains on other routes. For example, an examination of Fig. 4.1 reveals that if train 1 is moving through the junction from A to D it will block all trains on tracks C and B. However, it has no effect on trains travelling from D to A.

Two assumptions have been made when creating the simulator. The first is that trains are not allowed to enter the junction unless both the whole junction and the track section after the junction is clear. This prevents trains from sitting on the track section between the two junctions and causing gridlock. The second is that, to allow the trains to start at the specified distances from the junction, some of the track sections are very short. This means that in some cases the track sections are too short for a train travelling at maximum speed to have enough room to slow down in time if the next track section is blocked. Therefore, maximum speed limits on the short track sections are imposed which give enough time for the train with the smallest braking force to slow down in the distance available. The maximum speed limit through the actual junction is restricted to 64 km/h as specified in [2].

Table 4.2: The scheduled timetable for each train with delay penalties (based on [2])

Train Number	Train Type	Route	Delay Penalty (£/min)	Scheduled Arrival
1	Class 150	A to D	20	12:10
2	Class 220	D to A	40	12:12
3	Freight	B to C	10	12:19
4	Class 220	D to B	40	12:15
5	Freight	B to D	10	12:20
6	Class 150	D to B	20	12:19
7	Freight	A to C	10	12:28
8	Class 150	C to A	20	12:22
9	Class 220	C to A	40	12:27
10	Class 220	B to C	40	12:32
11	Freight	C to B	10	12:39
12	Class 150	A to D	20	12:36

Each 'tick', or movement of trains in the simulator, represents one second of time. The speed of the train at each time step is calculated using the Improved Euler Integration, also called Heun's Method. This allows the current acceleration and an estimate of the future acceleration to be combined to find the current speed of the train. Using this method allows for non-constant acceleration. The acceleration of a train at time t is calculated using Newton's Second Law of Motion ($F = ma$) and the power and resistance tables supplied by Kirkwood and Roberts [108], based on RailSys data. RailSys is used by Network Rail as a simulation tool [109]. Deceleration is a constant maximum brake force for each type of train as in Fan *et al.* [2].

As each train moves along its current track, it checks the status of its next track section. If the next track is occupied, the train will start to slow down when it reaches its stopping distance from the end of the track and will continue to slow down until it stops. If the track ahead is clear, the train will carry on, unless the track ahead has a lower speed limit than the current speed of the train, in which case the train will slow down until it reaches the required speed. If not slowing down or speeding up, a train will travel at its maximum speed or the maximum speed of its current track section, whichever is lower.

Table 4.2 shows the trains used, their routes through the junction, the penalty for delay and their scheduled arrival times. The delay penalties are different for

each type of train and are taken from Fan *et al.* [2]. The timetable was created by running all trains in numerical order through the simulator and recording their arrival times. This gave a base line measurement to be able to calculate the delay of the trains after a perturbation. Each train is one of three types: a Class 150 with a maximum running speed of 120 km/h, a Class 200 with a maximum running speed of 200 km/h, or a F2-mixed freight train with a maximum running speed of 110km/h [2]. Each type of train is of a different length, the Class 150 is 80.24 m long, the Class 220 is 187.4 m long and the F2-Freight train is 355 m long. The length of a train as well as its speed affects how long the train takes to clear its previous track section. This in turn determines how quickly the following train can move into the train's vacated track. Taking into account the length of the train is important as a train can occupy more than one track section at the same time. This is an improvement over the approach of Meng and Zhou [67] and Šemrov *et al.* [69] who, for simplicity, assume that a train can never occupy more than one track section simultaneously.

Dynamism was introduced to the simulator by adding a specified number of trains (m) at a specified time interval (f). The number of trains added represents the magnitude of change, and the time interval relates to the frequency of change. The new trains are created from the first m trains in the original timetable. The extra trains can be thought of as an extended timetable for the train junction and each combination of magnitude and frequency is run through the simulator in order to obtain the conflict-free timetable. All new trains are placed at the stations and are not allowed onto the track until the track section leaving the station is clear. At the point of change, any trains that are about to move into, or have moved into, the junction are removed by the simulator from the set of trains that need to be passed to the algorithm.

Fig. 4.3 shows a screen shot of the simulator. The text in the figure has been enlarged to make it easier to see. To be able to observe the movement of trains through the junction, the junction track sections are on a larger scale than the surrounding railway lines. The simulator was constructed using C++ Visual Studio 2012 with a graphical interface created using OpenGL. On a desktop computer with a dual core 3.2GHz processor and 6GB of RAM, it takes approximately 3 seconds to evaluate one sequencing solution.

4.1.4 The Extended Stenson Junction Train Simulator

For the second investigation, the simulator was extended to model the stations that feed into the junction as well as the junction itself. Each station corresponds to

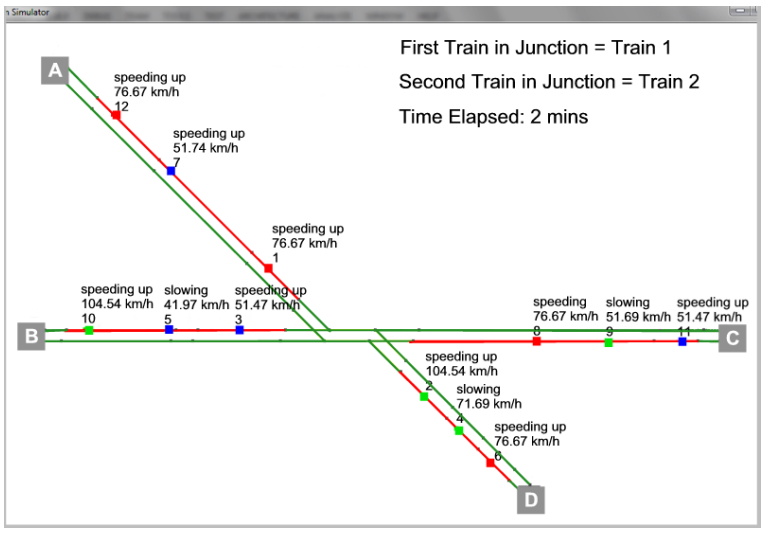


Figure 4.3: The junction simulator.

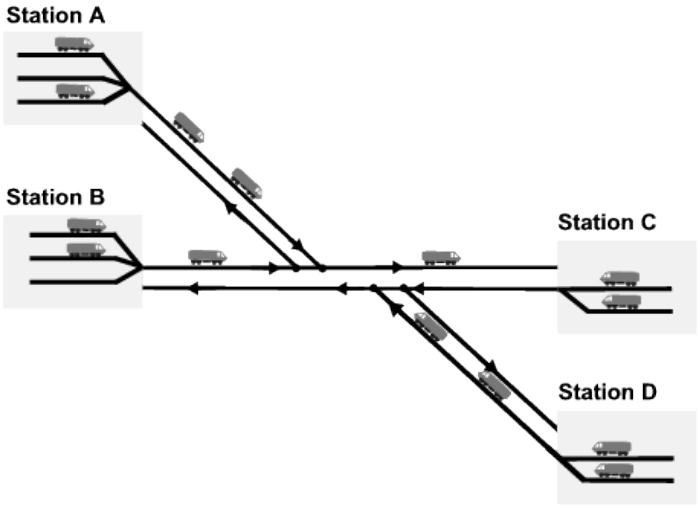


Figure 4.4: The Stenson Junction modelled with stations.

a real-world station on the British railway network. Station A is Crewe station; station B is Birmingham Station; station C is Derby station; and station D is Nottingham station. Crewe and Birmingham are much larger stations than Derby and Nottingham. Both Crewe and Birmingham have 12 platforms, Derby has 6 platforms and Nottingham has 7 platforms [110]. For this reason, stations A and B have been modelled with three exit platforms that feed into the junction while station C and D have been modelled with only two. Fig. 4.4 shows the stations with their platforms.

Modelling the section of the network in this way allows the algorithms to reschedule the trains at the stations as well as at the junction. More details about the

Table 4.3: The scheduled timetable for each train with delay penalties and station platforms

Train Number	Train Type	Scheduled Arrival	Station	Platform
1	Class 150	12:10	Crewe	1
2	Class 220	12:12	Nottingham	1
3	Freight	12:19	Birmingham	1
4	Class 220	12:15	Nottingham	2
5	Freight	12:20	Birmingham	2
6	Class 150	12:19	Nottingham	1
7	Freight	12:28	Crewe	2
8	Class 150	12:22	Derby	1
9	Class 220	12:27	Derby	2
10	Class 220	12:32	Birmingham	3
11	Freight	12:39	Derby	1
12	Class 150	12:36	Crewe	3

approach taken to allow them to do this can be found in Section 4.2.2.

Table 4.3 shows the trains, their scheduled arrival times and the platform they use at the station.

The following sections describe the ACO algorithms used in this investigation.

4.2 ACO for the DRJRP

As discussed in Chapter 3, ACO has never before been applied to dynamic railway rescheduling problems. However, the fact that it has previously been applied to static problems with good results [2, 23] suggests that it might be applicable to dynamic railway rescheduling problems. In addition, the fact that it has previously been applied to dynamic benchmark problems, such as the DTSP [24, 25, 26, 111, 27], indicates that it may be applicable to dynamic rescheduling problems. P-ACO was chosen for this first investigation, because it has an inbuilt memory with can allow potentially useful information from before the change to be retained and used in the next dynamic change period.

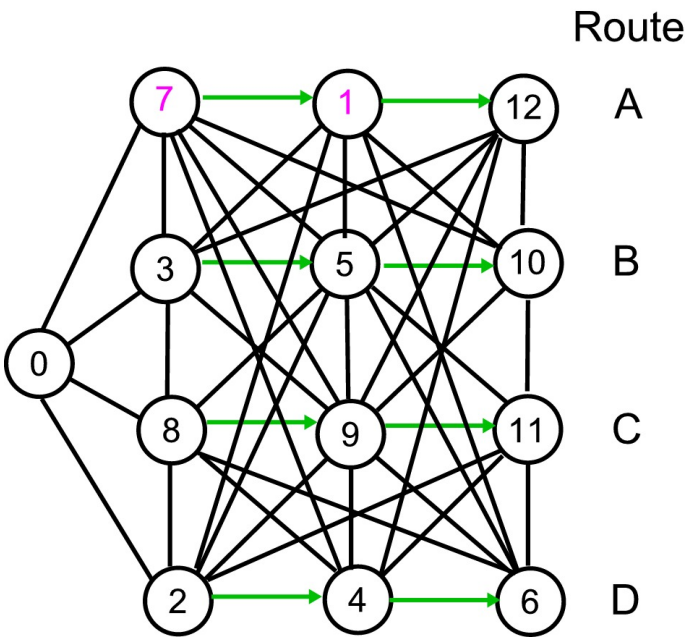


Figure 4.5: The initial directed edge graph.

4.2.1 Proposed P-ACO Algorithm for the DRJRP

In the first investigation, P-ACO was applied to the DRJRP. A description of P-ACO can be found in Section 2.2.2. In this implementation, the nodes that the ants visit represent trains that need to be rescheduled. The resulting ant tour is a list of trains in the order they are allowed to pass through the junction. The issue is how to represent the problem in a way that makes it possible for the ants to create their solutions while taking into account the fact that if the ants are allowed to visit any node in any order there is nothing to prevent them from creating infeasible solutions where a train is sequenced before the train in front of it.

To resolve this, the problem was decomposed into a fully connected, partly one-directional, weighted graph which has the ability to prevent an ant from making an infeasible tour (see Fig. 4.5). It is based on the design used by Van Der Zwaan and Marques [112] for the job-shop scheduling problem. At any one time, an ant only has the choice of one of the four trains sitting on the four access points into the junction. Each row in the graph represents one of the tracks that leads to the junction and is one-directional. This prevents an ant from choosing an infeasible train, for example, train 5 before train 3.

Node 0 represents the start node. At the beginning of each iteration, all ants are placed on this node. They then make a choice about which train node to choose next. After selecting a node, the next train on that train's track becomes visible to the ant and is included in its next decision. After all nodes have been selected, the

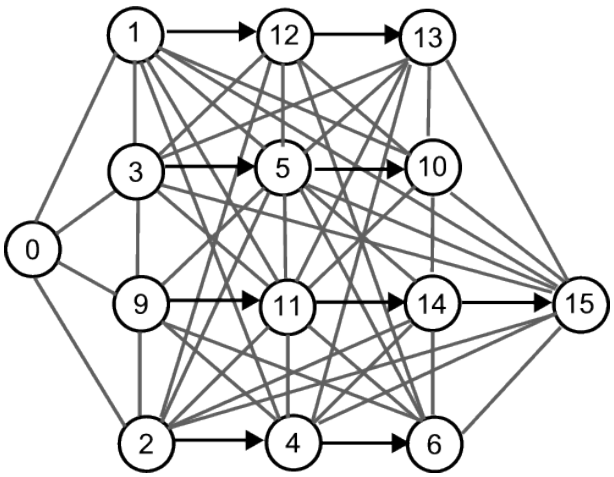


Figure 4.6: The problem after a change modelled by a fully connected, partly one-directional, weighted graph [1].

ant's tour is complete and the ant solution is evaluated by running it through the simulator. The best ant of the iteration is stored in memory and the pheromone is positively updated for that ant's tour.

In this implementation, the ants rely only on the pheromone values to guide them while making their choices and the value of β is set to zero. A computationally efficient and effective problem-specific heuristic could not be found. The natural choice for a heuristic would be the delay caused by sequencing each train. However, the delay of each train is dependent on the sequence of trains that went before it through the junction and is extremely difficult to establish as it changes for each ant's tour. An attempt was made to create an adaptive heuristic that builds the delay values as the ants sequence the trains, but this was extremely computationally expensive and performed worse than using the pheromone alone. The reason for the deterioration in performance is because the ants are unable to distinguish between the two types of delay present in the problem: the delay caused by the perturbation and the delay due to sequencing the trains in a particular order. An advantage of using only the pheromone values to guide the ants is that it reduces the amount of problem-specific knowledge needed to run the algorithm.

After a change, the graph may shrink or grow depending on the number of trains added and the number of trains removed from the problem because they have passed through the junction and are no longer relevant. Fig. 4.6 shows the graph after a dynamic change. Train 8 has been removed from the graph because it has passed through the junction and is no longer available for resequencing. Trains 14 and 15 have arrived on route C and have been added to the graph on the line that corresponds to route C.

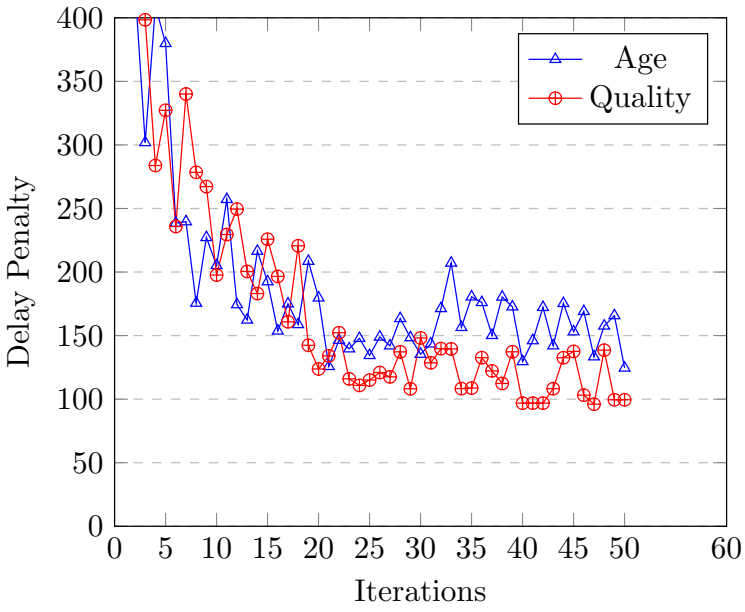


Figure 4.7: A comparison of ant removal strategies.

Ant Removal Strategy

The fact that the algorithm has a memory raises the question of which ant to remove from the memory once it reaches its pre-set capacity. An experiment carried out to establish whether to remove the oldest ant (Age Strategy) or the worst ant (Quality Strategy) from the memory revealed that removing the worst ant gave slightly better performance over removing the oldest ant. Fig. 4.7 shows the delay penalty for both the Age and Quality strategies averaged over 10 runs of the algorithm.

This is in accordance with the research carried out by Guntsch and Middendorf [36], who found that, in a problem where the ants rely purely on the pheromone to guide them, removing the worst ant solution from the memory results in a good performance; possibly because it provides strong and fairly consistent guidance. This strategy also ensures that the best ant of all the iterations is retained in the memory and provides the elitism that Guntsch and Middendorf [36] found beneficial when running P-ACO without a heuristic. They suggested that the elitism goes some way to providing the guidance that is missing due to the lack of a heuristic.

Repairing Memory After a Dynamic Change

To make use of the information held by the ant solutions in memory, the ants' tours have to be repaired. This is achieved by removing any train in an ant's tour that has been removed by the train simulator because it is about to pass into, or has passed into, the junction, and adding the new trains to the end of the solution in

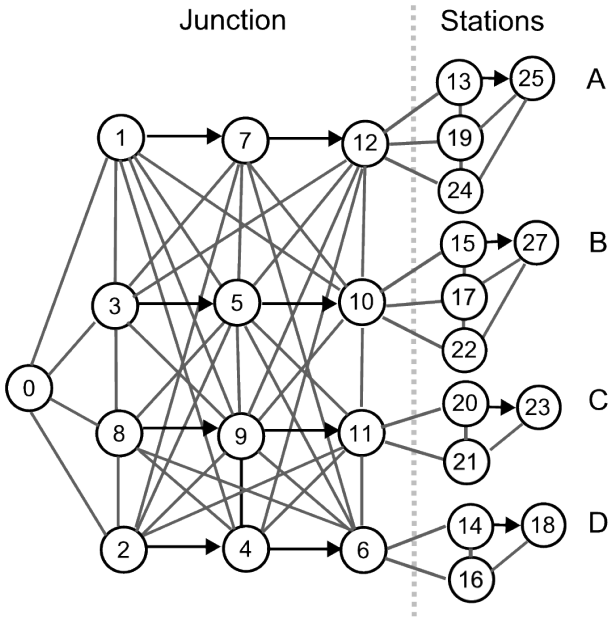


Figure 4.8: The directed edge graphs for the junction and associated stations.

the order dictated by the train timetable. This is similar to the KeepElitist strategy used by Guntsch and Middendorf [24] for the DTSP, where cities that are no longer present in the environment are removed and new cities are placed where they cause minimum increase in the distance between cities.

4.2.2 P-ACO for the Extended DRJRP

The directed edge graph described in Sec. 4.2.1 only allows the ants to resequence the trains through the junction. It is proposed to also allow the ants to resequence the trains at the stations in the hope that this will allow them to resolve the delays introduced at the stations. This is also seen as a first step in extending the algorithms to be able to deal with a much larger area of the railway network.

The stations that feed into the junction have been modelled within the simulator and each train has been allocated to a platform at the station. The original directed edge graph has to be updated to reflect this change. Fig. 4.8 shows that the problem has now been modelled as several linked graphs: One graph for the junction and one for each station. Each horizontal line in the station graphs represents a platform at the station. As previously mentioned, stations A and B, being much larger stations, have been modelled as having three platforms while stations C and D have only two platforms. Once the ant gets to the end of a horizontal line on the junction section graph, it has access to the graph that represents the trains at the station. An ant can only choose trains in the order that they arrive at the platform as denoted by

the one-way arrow. This constraint ensures ants can only make feasible schedules and removes the possibility of trains having to pass over other trains waiting before them on the same platform. As before, the shape of the graphs changes over time as new trains are added and trains that have passed through the junction are removed.

In the version of the algorithm that does not sequence the trains at the stations, denoted Non Station Sequencing P-ACO (NSS-PACO), trains arriving at the stations are simply added onto the back of the graph as shown in Fig. 4.6.

Adapting the Memory in P-ACO After a Change

In the extended DRJRP, the fact that ants are allowed to also resequence trains at the stations creates a problem for P-ACO as it becomes extremely difficult to repair the ants after a change. This is because the algorithm can change the arrival order of the trains at the junction. This means the solutions held in memory after a change may contain trains with different arrival orders to the solution used to make the snapshot of the junction at the point of the dynamic change (see Section 4.3.1). In effect, the memory now contains infeasible solutions. This makes it extremely difficult to repair the ants in memory by just removing the trains that have passed through the junction and adding in the new trains. The arrival order of the trains in the solutions also needs to be amended, which would involve reshuffling the order of all the trains in the solutions. Performing this reshuffle would require extreme domain knowledge and would result in solutions so different from the original solution that the information they contained would be lost.

To get around this problem, it is proposed, in the algorithms that are able to schedule the trains at the stations, to discard all the ants in memory after a change and replace them with new solutions or immigrants. The immigrants can be either elite immigrants, based on the best ever ant, random immigrants, or a mixture of elite and random immigrants. The algorithms that use these schemes are labelled EI-PACO, RI-PACO and HI-PACO respectively.

For the algorithm that does not resequence trains at the stations (NSS-PACO), the ants are repaired as in the non-extended DRJRP. Any train in an ant's tour that is no longer of relevance to the problem is removed and new trains are added to the end of the tour in the order dictated by the train timetable.

Using Immigrants for DOPs

The aim of an EC approach is to converge to an optimum solution. However, this is an issue if the problem is a dynamic one as there may not be enough diversity in the population to allow the algorithm to adequately explore the search space after

a change in the problem. To help solve this issue, Grefenstette [113] introduced the idea of adding random solutions, named immigrants, to the population to maintain diversity and to give the algorithm the flexibility to adapt to a changing environment.

Grefenstette found that adding random solutions to a population improved the performance of a GA on DOPs [113]. Later work [114] found that basing the immigrants on existing individuals that have been found to perform well in the new environment efficiently improved the performance of GAs for DOPs, while basing the immigrants on the best solution of the current generation worked well particularly when the changes in the environment were slow and slight [115, 116].

This idea has been extended to ACO algorithms on the DTSP where the problem was made dynamic by removing and adding cities [111]. In this work, at every iteration, a proportion of the worst ants were replaced with either random or elite immigrants or a hybrid mix of the two. This removed the need, in this DTSP problem, for the expensive and domain specific repair of the solutions in memory that is needed by the P-ACO algorithm after a change in the environment. The addition of immigrants was found to significantly improve the performance of the P-ACO algorithm with the elite immigrants performing significantly better than random immigrants on slow changing environments and the random immigrants performing slightly better than elite immigrants on most fast changing environments. The hybrid immigrants were found to significantly outperform both the random and elite immigrants.

Mavrovouniotis and Yang [27] also investigated the addition of immigrants schemes to the DTSP with traffic factors, where the distances between cities change over time in either a random or cyclic pattern. Again, the immigrants were found to provide significant improvement over the algorithms without immigrants schemes.

The above findings suggest that incorporating immigrants into the ACO algorithms for DOPs is not only possible but also effective.

The random, elite and hybrid immigrants used in this work are described in more detail below.

Random Immigrants

Random immigrants are made by constructing solutions where the ants randomly choose the next feasible train node with no regard for the pheromone trails. This results in an ant with a feasible solution that does not represent the existing pheromone trails in any way. Random immigrants have the advantage of being able to inject diversity into the population but may result in the loss of information.

Elite Immigrants

The elite immigrants are based on the best ant so far. This makes sense in terms of the operation of the algorithm as at each change the best solution for that change period is chosen to run the simulator to provide the snapshot of the state of the junction to pass to the algorithm (see Section 4.3.1). Care must be taken when the immigrants are constructed. It is not enough to simply swap around trains in the best ant's solutions as this would very quickly result in infeasible solutions where trains would be scheduled in front of trains that precede them on the track.

Previous research has created elite immigrants using the adaptive inver-over operator [111, 117]. However, this algorithm does not preserve the order of the trains in the solution and would have a high probability of creating infeasible solutions.

What is required is a way to generate new solutions from the best ant while preserving the order of trains on each of the routes into the junction. In this paper, this was achieved by using a path-preserving local search heuristic to create feasible solutions from the current best solution. The heuristic used is based on the *lpp-3-exchange* search procedure used by Gambardella and Dorigo [118]. The local search algorithm as applied to this problem is described below.

Path-Preserving Local Search

In order to explore all possible feasible combinations that can be made from a solution, the path preserving local search heuristic makes two passes through the solution vector; a forward pass and a backwards pass.

Forward Pass

This part of the search starts at the beginning of the train sequence and looks for feasible swaps between the first train and the subsequent trains. To explain, consider a feasible train order <3,8,9,5,11,2,4,7,6,1,10,12>.

The first swap would be between train <3> and, the next train in the sequence, i.e., train <8>. Swapping these two trains would have no effect on the feasibility of the solution because they are on different routes into the junction (see Fig. 4.1). It would result in the feasible train sequence <8,3,9,5,11,2,4,7,6,1,10,12>. No train in this sequence arrives before any other train on the same route and therefore this solution would make a feasible elite immigrant.

We then consider a swap between train <3> and trains <8,9> in the original train sequence. This comparison would involve swapping train <3> with trains $< 8, 9 >$ to give the solution <8,9,3,5,11,2,4,7,6,1,10,12>. Again, this is a feasible

sequence as even though 8 and 9 are both on route C into the junction, their order is preserved and so the solution is feasible. In addition, trains 8 and 9 are on route C whereas train 3 is on route B, which means that placing trains 8 and 9 before train 3 does not produce a situation where one train would have to pass through another on the same route to get to the junction.

The next step is to consider a swap between train <3> and trains <8,9,5>. However, this causes a problem as a swap between these trains results in the solution <8,9,5,3,11,2,4,7,6,1,10,12>. In this case, train 5 is now before train 3 in the sequence and therefore train 5 would be required to pass through train 3 in order to reach the junction, which is a physical impossibility. Once an infeasible solution is reached there is no point continuing with any more comparisons using train 3, as every other combination would mean that train 5 would have to pass through the junction before train 3.

At this point, the search for feasible swaps would move onto evaluating swaps between the first two trains in the sequence and the rest of the trains in the sequence, that is, trains <3,8> with train <9>, then train <3,8> with trains <9,5> etc. Again, once an infeasible swap is detected, the search using <3,8> is halted and the search moves onto evaluating swaps between trains <3,8,9> and the rest of the train sequence.

The process continues until the evaluation of the final swap between trains <3,8,9,5,11,2,4,7,6,1,10> and train <12>. After this swap evaluation, a backward pass, using a similar search procedure, is performed on the same sequence of trains.

Backward Pass

The backward pass is similar to the forward pass but starts at the end of the sequence of trains. It first considers a swap between train <12> and train <10> to make the feasible solution <3,8,9,5,11,2,4,7,6,1,12,10>. It then evaluates the feasibility of swapping train <12> with trains <1,10> to give the sequence <3,8,9,5,11,2,4,7,6,12,1,10>. This is an infeasible sequence as train 12 cannot pass through the junction before train 1. The search for feasible solutions with train <12> is halted and the search for feasible swaps moves onto the next two trains in the sequence, trains <10,12>. Again, this process continues until the final comparison of trains <8,9,5,11,2,4,7,6,1,10,12> with train <3>.

Carrying out the search for feasible solutions in this way has the advantage of cutting down on the number of evaluations needed as the procedure is halted as soon as an infeasible solution is found. In addition, in this particular train sequencing problem, the process of determining if a swap is feasible is a relatively quick one as

it only needs to be performed on trains that are on the same route.

The elite immigrants are made by using the above technique, to find all the feasible local search solutions, and randomly choosing one of those solutions to become the immigrant. This means that the elite solutions are very similar to the best solution used to make the snapshot of the junction after a dynamic change and allow pertinent information to be carried over to the new environment.

Hybrid Immigrants

There is a danger with elite immigrants in that their exploitation of a good solution could limit the algorithm's exploration of the search space. For this reason, a third immigrant scheme (HI-PACO) is also investigated where the immigrants introduced consist of a half and half mix of elite and random immigrants. The hope is that the addition of random immigrants will encourage the ants to explore the search space whereas the elite immigrants will allow the ants to exploit the previously found good solution.

In EI-PACO, RI-PACO and HI-PACO algorithms, the pheromone matrix is reinitialised after a dynamic change and updated with the solutions of the new ants in memory. If only elite immigrants are used, this will give a pheromone trail that is based on the solution used to create the snapshot of the simulation before a change. This will encourage exploitation of this solution but may have the disadvantage of reducing diversity. However, if random immigrants are used, then this is in effect a restart of the algorithm and will result in the loss of any information from before the change. However, they will have the advantage of introducing diversity into the algorithm, which will encourage the exploration of the search space. Hybrid immigrants may be able to combine the advantages of both these approaches by making use of information before the change but also introducing diversity to encourage further exploration of the search space.

This research takes a slightly different approach to [111, 27] as instead of adding immigrants at every iteration they are only added after a change to equip the algorithm for the new environment.

4.2.3 Proposed MMAS Algorithm for the Extended DRJRP

In the second investigation based on the extended DRJRP a second ACO algorithm, MMAS, was applied to the problem. MMAS has a different mechanism for adapting to change than P-ACO (see Section 2.2.3). In this case, instead of a memory that

is retained between changes, MMAS holds a memory of previous good solutions in the pheromone trails it retains between changes. This means that MMAS has no need of a repair operation to repair the memory between changes. The directed edge graph the ants use to make their solutions is the same as that used by P-ACO (see Section 4.2.2).

In this implementation the ant used to update the pheromones is chosen in a ratio of 10:1 of the best-so-far ant to the best iteration ant. For 10 iterations, the best-so-far ant is chosen followed by one iteration when the best-iteration ant is chosen. This was found to give the best performance in preliminary investigations.

4.3 Experimental Design

The following sections elaborate on the experimental design details of this work, including the dynamics implementation, the handling of constraints, the limitations of the model and the performance measure.

4.3.1 Dynamics Implementation

Even though the trains and junctions are simulated, this is a real-world problem and requires consideration of how it could be implemented in a real-world delayed-train scenario. The supposition is that after a perturbation, the algorithm is run very quickly in parallel on several computers in order to find a solution as near optimal as possible in the time available. Ant algorithms are very suitable for running in parallel [22] and doing so would allow a solution to be found within the maximum rescheduling time of 180 s suggested by the French operating company SNCF [58]

The sequence of trains in the best solution is then run through the junction until the dynamic change occurs. This change is triggered by the arrival of more timetabled trains. At the point of change, a 'snapshot' is taken of the junctions by the simulator. This records the status of the trains, track and junction at that moment in time. The snapshot, plus the new trains, is passed to the P-ACO algorithm, and the algorithm is run again to find the best solution for the new environment. In this way, the algorithm and the simulator are very loosely coupled. The algorithm only acts on the information given to it and does not influence the simulator in any way. This has the advantage that both the simulator and the algorithm can be modified independently of each other.

When the algorithm receives the updated train information, it reconstructs the directed edge graph to reflect the trains in the simulator snapshot. Any trains that

have been removed from the snapshot are also removed from the graph and node 0 is reconnected to the next four trains sitting at the junction.

4.3.2 Handling Constraints

One of the constraints of the extended DRJRP is that a train is not allowed to enter the junction before the train in front of it on the same track, and is not allowed to leave the station before the train in front of it on the same platform. This constraint is handled by creating the directed edge graph in such a way that ants can only make a decision as to which train to sequence next from the set of trains that are waiting at the start of the junction or are waiting at the exit of the station (see Section 4.2.1). This graph structure prevents infeasible solutions from being created by the ants and is more computationally efficient than allowing the ants to make unrestricted solutions and then having to identify and discard all the infeasible solutions.

Further constraints of the problem are that trains are not allowed to enter a block section occupied by another train and that trains are not allowed to cross the path of other trains entering or leaving the junction. Both these constraints are dealt with in the Stenson junction simulator. In the first case, a train can only enter the next track section if it is clear of all other trains, and in the second case a simulated interlocking system, incorporated within the simulator, prevents trains from crossing the path of other trains.

4.3.3 Limitations of the Model

The present model is limited by the fact that, although stations have been included to extend the model, it still considers only a small area of the British railway network. As the simulator works at the microscopic level of block sections, extending the model to a larger area would increase the computational complexity of the problem and could mean that, in the case of the ACO algorithms, more ants may be needed to obtain the same results.

4.3.4 Performance Measure

In an ideal world, the optimal solution would be available in advance to allow the effectiveness of the algorithm to be evaluated. Establishing the optimum would involve some form of brute force algorithm. There are 369,600 feasible sequences of 12 trains, on four routes, that can be created as possible solutions to the static problem [2]. Each solution takes approximately 3 seconds to evaluate, on a single core Xeon Woodcrest Linux processor running at 2.83GHz, which means the static

problem alone would take approximately 12.83 days to evaluate all the possible solutions to find the optimal solution. The additional trains added in the dynamic problem would increase the number of possible feasible solutions and would give an exponential increase in the time needed to find the optimal solutions.

For this reason, the performance of the proposed algorithm is instead evaluated against a FCFS algorithm where trains are assigned to the junction in the order that they arrive at the junction. This performance measure is commonly used by railway controllers to reschedule trains after a perturbation [32] and has previously been used by [2] and [49] to evaluate train rescheduling algorithms.

4.4 Experimental Investigation 1

In this first experimental study the performance of the proposed P-ACO algorithm is compared to the performance of FCFS on the DRJRP. In this case only the junction is taken into account, the ants do not resequence the trains at the station.

4.4.1 Experimental Settings

The following pheromone parameters were implemented, as recommended by Guntsch and Middendorf [36]. The maximum pheromone value (τ_{max}) was set to 1, the minimum pheromone value (τ_{init}) was set to $1/n$, where n is the number of nodes, and the pheromone update value to $(\tau_{max} - \tau_{init})/k$, where k is the size of the memory. All pheromone levels were initialised to τ_{init}.

The other parameters were established by preliminary experimentation. The best combination was found to be 12 ants with a memory size of 6 and a q_0 value of 0.1. After 150 iterations, very little improvement was found to occur in the ants' solutions. Therefore, the algorithm was run for 150 iterations before each dynamic change.

4.4.2 Experimental Results

Nine different dynamic environments were investigated involving all permutations of three different magnitudes of change (2 trains, 5 trains, 8 trains) and three different change frequencies (5 mins, 10 mins, 15 mins). For both the P-ACO and FCFS algorithms the total delay penalty at the point of change was recorded. After the last dynamic change, the algorithm was run for a further 150 iterations and the delay penalty was recorded at the end of the iteration period. The total delay penalty recorded did not include the delay of any trains that had been removed

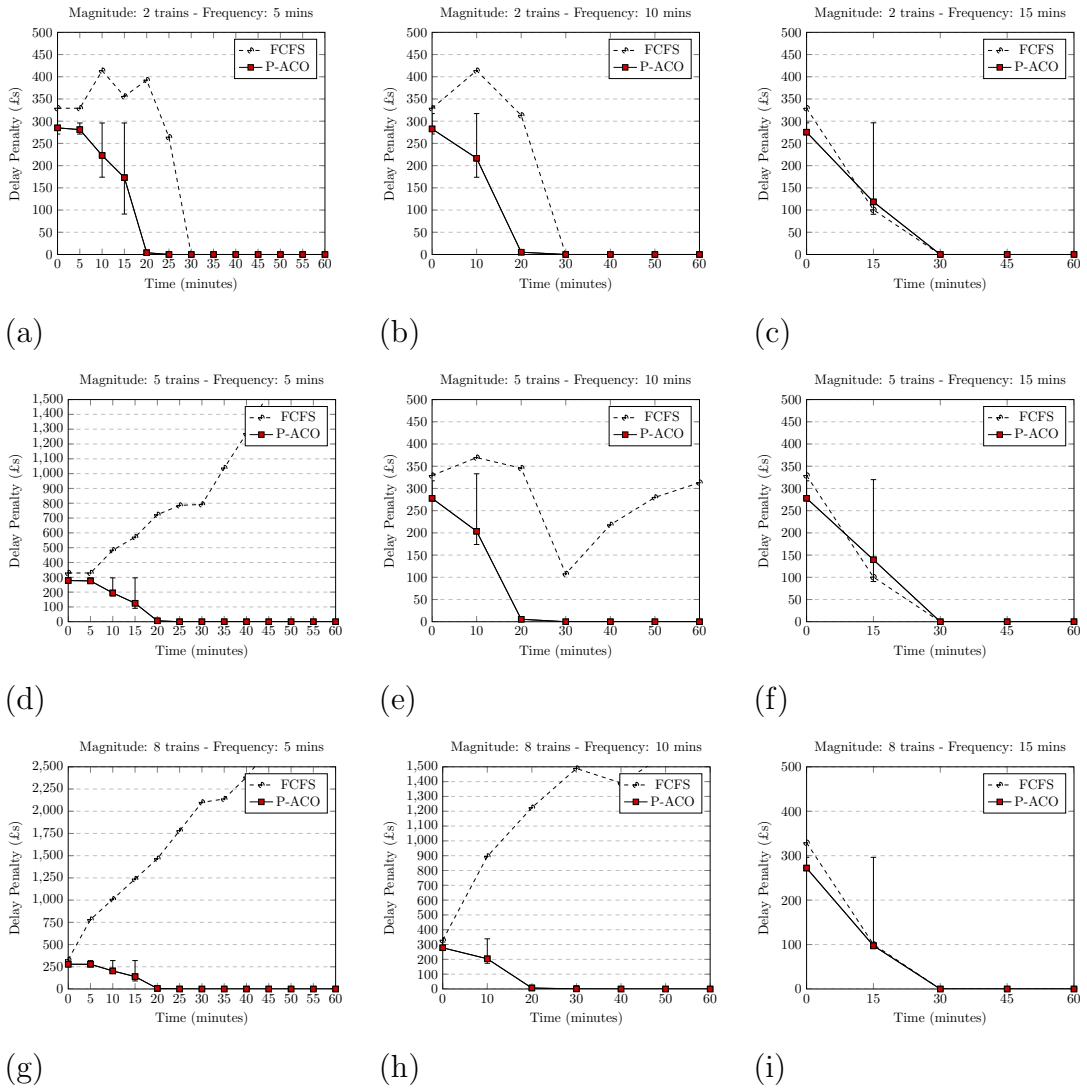


Figure 4.9: A comparison between the performance of FCFS and P-ACO on each of the dynamic rescheduling problems for Scenario 2

from the consideration by the algorithms because they were about to pass through or had passed through the junction. Thirty runs were completed for each dynamic environment and the results averaged. Fig. 4.9 shows the outcome for each of the nine combinations of magnitude and frequency. The dashed line represents the delay penalty using FCFS while the unbroken line represents the delay penalty using P-ACO. The vertical lines indicate the maximum and minimum delay penalties produced by the P-ACO algorithm to give an indication of variance. The scale of the different graphs varies to accommodate the maximum delay penalty.

It is apparent from the results that P-ACO outperforms FCFS in all cases where the frequency of change is high, irrespective of the magnitude of change. However,

Table 4.4: Summary of the algorithms investigated

Abbreviation	Algorithm Description	Station Sequencing
EI-PACO	P-ACO where the memory is replaced with repaired elite immigrants after a change	Yes
RI-PACO	P-ACO where the memory is replaced with repaired random immigrants after a change	Yes
HI-PACO	P-ACO where the memory is replaced with repaired elite and random immigrants after a change	Yes
NSS-PACO	P-ACO where the ants in memory are retained and repaired with KeepElitist repair operation	No
MMAS	The Max-Min AS algorithm	Yes

the largest improvement in performance is seen when the dynamic change is not only of a high frequency but also of a high magnitude. For example, when 8 trains are added every 5 minutes (Fig. 7(g)), the average delay penalty is £277.99 for P-ACO and £781.33 for FCFS after the first change, and is £203.37 for P-ACO and £1009.67 for FCFS after the second change. In addition, after 20 minutes of changes, the effect of the perturbation caused by train 7 arriving before train 1 has been mitigated by P-ACO but persists for FCFS and continues to increase. In the case of low frequency changes (Figs. 7(c), (f) and (i)), the difference between P-ACO and FCFS is minimal. This is in line with the findings of Corman *et al.* [42] who found that on low volume traffic areas simple algorithms perform adequately while in high density traffic situations advanced scheduling algorithms improved performance.

4.5 Experimental Investigation 2

In this second investigation, an experimental study is carried out to compare the ability of five ACO algorithms to optimise the extended DRJRP with multiple delays over time. The investigated algorithms are EI-PACO, RI-PACO, HI-PACO, NSS-PACO and MMAS. A breakdown of the characteristics of each of these algorithms can be found in Table 4.4. The performance of these algorithms was also compared to the solutions produced using FCFS.

4.5.1 Experimental Settings

For all the P-ACO algorithms, the same parameters were implemented as in investigation 1 (see Section 4.4.1).

For MMAS, experimental investigation found that the best parameters were $a = 20$, $p = 0.5$, 14 ants and pheromone reinitialisation when there is no change in the best solution for 30 iterations. In all cases, the algorithms were run for 150 iterations before a dynamic change.

In the case of EI-PACO and RI-PACO, six immigrants were used to replace the memory after a dynamic change, whereas in HI-PACO three elite immigrants and three hybrid immigrants were used.

4.5.2 Experiment Results

As previously mentioned in Section 4.1.2, extra delays over the course of the dynamic problem can only be introduced if the number of trains added to the simulation is of a sufficiently high number to make it feasible to swap the arrival times of the trains at the stations. Therefore, in all the following experiments, the magnitude of dynamic change (m) is eight. The addition of eight trains creates a situation at station D where the arrival order of two trains at the station can be swapped (see Table 4.1) and a delay introduced. Either one or two additional delays were added at the stations for each of the three different change frequencies (5 mins, 10 mins, 15 mins). This made a total of six dynamic scenarios.

Thirty runs were completed for each dynamic environment. The graphs in Fig. 4.10 show the results for each of the change frequencies when one additional delay was introduced to the scenario at change 1, i.e., after 5, 10 or 15 minutes of running the simulation. The graphs in Fig. 4.11 show the results of adding two additional delays, one at change 1 and the second at change 4. The x-axis is the time passed in minutes between each dynamic change, the y-axis is the average fitness of the algorithm in terms of the delay penalty. The scale of the different graphs varies to accommodate the maximum delay penalty.

From the graphs (see Fig. 4.10 and Fig. 4.11), it is apparent that the addition of extra delays over the period of investigation appears to have the most marked effect when the frequency of dynamic change is high. When the dynamic change is low or moderate, most algorithms appear to be able to resequence the trains to mitigate the effect of the multiple delays. This suggests that, in a very busy section of the railway track, delays have a more pronounced effect than in a less busy section. Nevertheless, EI-PACO, HI-PACO and NSS-PACO managed to efficiently remove

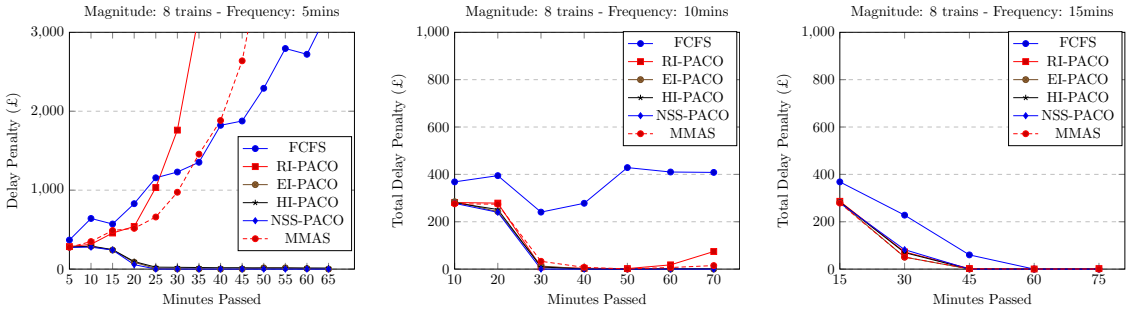


Figure 4.10: Experimental results for each of the algorithms for different frequencies of dynamic change and one additional delay at station D.

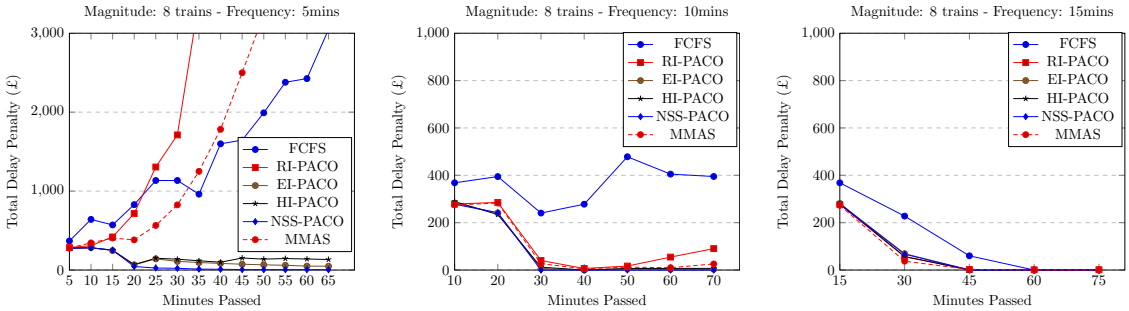


Figure 4.11: Experimental results for each of the algorithms for different frequencies of dynamic change and two additional delays at station D.

or mitigate the delay in both the one-delay and two-delay scenarios.

Results were tested for statistical significance using the Kruskal-Wallis test for multiple comparisons followed by the Wilcoxon rank-sum, non-parametric, pairwise test with Bonferroni correction at a 0.05 significance level. FCFS was compared to the ACO algorithms using the one-sample Wilcoxon signed rank test as in this case a single result is being compared with multiple results for the ACO algorithm. Results are shown in Table 4.5. This table shows the results of comparing Algorithm1 $\Leftrightarrow$ Algorithm2, where the symbol '$s+$' indicates that Algorithm1 is significantly better than Algorithm2 while '$s-$' indicates that Algorithm1 is significantly worse than Algorithm2, and the symbol '$\sim$' indicates no significant difference between the two algorithms.

The results show that EI-PACO, HI-PACO and NSS-PACO all significantly outperform FCFS on all the delay scenarios. This is in line with the findings of investigation 1 which showed that P-ACO outperformed FCFS on all the high magnitude scenarios. In the extended DRJRP, the magnitude of dynamic change (m) is always high (see Section 4.5.2).

Table 4.5: Non-parametric statistical results of comparing the algorithms on different delay scenarios at 0.05 significance level

	1 Station Delay			2 Station Delays		
Algorithms	$f \Rightarrow 5$	10	15	$f \Rightarrow 5$	10	15
RI-PACO $\Leftrightarrow$ FCFS	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$
EI-PACO $\Leftrightarrow$ FCFS	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
HI-PACO $\Leftrightarrow$ FCFS	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
NSS-PACO $\Leftrightarrow$ FCFS	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MMAS $\Leftrightarrow$ FCFS	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$
RI-PACO $\Leftrightarrow$ EI-PACO	$s-$	$s-$	$\sim$	$s-$	$s-$	$\sim$
RI-PACO $\Leftrightarrow$ HI-PACO	$s-$	$s-$	$\sim$	$s-$	$s-$	$\sim$
RI-PACO $\Leftrightarrow$ MMAS	$s-$	$\sim$	$\sim$	$s-$	$\sim$	$\sim$
EI-PACO $\Leftrightarrow$ HI-PACO	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
EI-PACO $\Leftrightarrow$ MMAS	$s+$	$\sim$	$\sim$	$s+$	$s+$	$\sim$
HI-PACO $\Leftrightarrow$ MMAS	$s+$	$\sim$	$\sim$	$s+$	$s+$	$\sim$
RI-PACO $\Leftrightarrow$ NSS-PACO	$s-$	$s-$	$\sim$	$s-$	$s-$	$s+$
EI-PACO $\Leftrightarrow$ NSS-PACO	$\sim$	$\sim$	$s+$	$s-$	$\sim$	$s+$
HI-PACO $\Leftrightarrow$ NSS-PACO	$s-$	$\sim$	$\sim$	$s-$	$\sim$	$s+$
MMAS $\Leftrightarrow$ NSS-PACO	$s-$	$s-$	$s+$	$s-$	$s-$	$s+$

However, RI-PACO performs significantly worse than FCFS on the high magnitude high frequency changes ($m = 8, f = 5$) for both the single station delay and double station delay. This is most likely because using random immigrants to initialise the pheromone trails after a change is effectively restarting the algorithm from scratch. All the information from the environment before the change is lost. In a situation when trains are arriving at high frequency, this lack of information from the previous environment seriously hampers the ability of the algorithm to find an effective solution. In such a scenario, trains are added more quickly than trains are lost from the problem, which means that the size of the problem, and the size of the search space, increases after each dynamic change. Restarting the algorithm with such a large search space has the consequence that the ants may not be able to explore it sufficiently in the time available and may be unable to find a good solution. When the frequency of changes is low, the time between the addition of new trains to the problem increases, which means that more trains will have had time to pass through the junction. This will result in less trains for the ants to sequence, with a consequently smaller search space. In this case, even when restarting the algorithm from scratch, the ants are able to explore the whole search space to find an optimal solution. This indicates that in the extended DRJRP, when the search space is large, knowledge carried over from previous environments is especially valuable.

MMAS performs worse than FCFS when the change is of high magnitude and high frequency ($m = 8, f = 5$) and significantly better than FCFS for the medium and low frequency changes, for both the single station and double station delay scenarios. This is because MMAS uses only the pheromone evaporation to remove trails that are no longer relevant to the new environment and in addition does not have the advantage of repaired solutions from the previous environment to initialise the amended pheromone matrix and to guide it after a dynamic change. Higher frequency train additions mean a larger search space and the algorithm may struggle to explore it effectively without any guidance. Low frequency train additions mean that the search space is smaller and the algorithm is able to explore it effectively and to find an optimal solution.

RI-PACO performed significantly worse than EI-PACO on the high and medium frequency changes. This is because basing the immigrants on the solution used to make the snapshot of the simulation for the next change period means that the solutions are better suited to the new environment than randomly created ants. This may be a peculiarity of this real-world scenario. Due to the need to keep the trains running during the period of disruption, a decision has to be made as to the best solution to choose to run the trains for the next dynamic change period. Thus,

the new environment is influenced by the solution used to run the trains for the next change period and immigrants based on that solution have an advantage in the new environment.

RI-PACO also performs significantly worse than HI-PACO on the high and medium frequency changes. This is most likely because the addition of the elite immigrants to the HI-PACO algorithm means that there are some ants well suited to the new environment which helps the search. In fact, RI-PACO appears to be one of the worst performing algorithms; it is also outperformed by MMAS and NSS-PACO on the high and medium frequency scenarios. In this dynamic problem, adding random immigrants to the memory after a dynamic change does not appear to be an effective solution.

There is no significant difference between the performance of EI-PACO and HI-PACO in all scenarios, which indicates that in this dynamic problem the addition of the random immigrants to the elite immigrants does not significantly improve the performance of the algorithm. This is most likely again because of the nature of the problem. The fact that the best solution is used to make the snapshot for the next change period means that the environment after the change is similar to the environment before the change and therefore what is required is the exploitative power of the elite immigrants rather than the disruption introduced by the random immigrants.

MMAS performed significantly worse than EI-PACO and HI-PACO on the high frequency scenarios with one delay and the high and medium frequency scenarios with two delays. Again, this is most likely because MMAS does not have any mechanism to cope with dynamic change apart from the evaporation of redundant pheromone trails and this is not as effective as P-ACO's method of coping with dynamic change by completely removing redundant trails. MMAS's lack of ability to cope with dynamic change is the most detrimental when the changes occur with high frequency because the algorithm may not have sufficient time to find a good solution before the next change occurs.

Finally, although it was believed that adding the capability to sequence trains at the stations would improve the algorithms' ability to find a solution to the extended DRJRP, this has not been found to be the case. NSS-PACO outperformed all algorithms on almost all the delay scenarios where the changes occurred with high (every 5 min) or medium (every 10 min) frequency. However, NSS-PACO was outperformed on the low frequency changes for both the single and double station delay scenarios. Low frequency changes mean a smaller search space which suggests that the failure of the station sequencing algorithms to perform as well as the

non-station sequencing algorithms may be because station sequencing results in an extended search space with more choices for the ants. When the search space is small, sequencing at the stations may offer a slight improvement. This suggests that the number of ants used may not have been enough to cope with the increase in the size of the search space and additional mechanisms may be necessary to deal with this larger search area.

4.5.3 Algorithm Computation Time

The algorithms were executed on a single core Xeon Woodcrest Linux processor running at 2.83GHz. The computation times varied according to the number of trains in the problem at a given change. The reason for this is that over 99% of the computation time was taken up by the time taken to evaluate the ants' solutions in the simulator. The more trains in the problem, the longer the evaluation process. All the investigated ACO algorithms performed similarly in terms of execution time. Therefore, to illustrate the computation time, the execution time of just one of the algorithms, EI-PACO is given. Table 4.6 shows the computation times for this algorithm when 8 trains are added every 15 minutes, while Table 4.7 shows the computation times when 8 trains are added every 5 minutes. In the tables, the first, second and third lines are the average time, the minimum time and the maximum time over all 30 runs. The last line of the tables shows the percentage of computation time that was taken up by evaluating the ants' solutions in the simulator.

Adding 8 trains every 15 minutes means that many trains have passed out of the problem before the new trains arrive at each change. Therefore, the average execution time does not increase greatly over the course of the changes. For example, for change 1 the average execution time for 150 iterations is 1 minute 26 seconds (0.57 seconds per iteration) while at change 4 the average execution time is 2 minutes 6 seconds (0.84 seconds per iteration) (see Table 4.6).

However, when 8 trains are added every 5 minutes, the average execution time at change 1 is 1 minute 58 seconds (0.79 seconds per iteration), at change 4 it is 5 minutes and 4 seconds (2.03 seconds per iteration) while at change 12, because of the large number of trains in the system, it has risen to an average of 18 minutes and 52 seconds (7.55 seconds per iteration) (see Table 4.7).

This execution time is, of course, unacceptable in a real world situation. However, ACO is very amenable to being run in parallel [22], which would cut down the computation time considerably and would make it feasible for real-time operation. In addition, the algorithm could be run for less iterations by choosing a different termination criterion, for example, running the algorithm until there has been no

Table 4.6: Algorithm execution times in **minutes** for EI-PACO with $m = 8$ and $f = 15$

Change	1	2	3	4
Average	1.43	1.47	1.84	2.10
Minimum	1.36	1.37	1.63	1.85
Maximum	1.49	1.66	2.08	2.52
Evaluation	99.90%	99.90%	99.92%	99.92%

Table 4.7: Algorithm execution times in **minutes** for EI-PACO with $m = 8$ and $f = 5$

Change	1	2	3	4	12
Average	1.97	2.89	3.96	5.07	18.87
Minimum	1.93	2.77	3.77	4.81	17.16
Maximum	2.02	3.03	4.25	5.47	34.82
Evaluation	99.91%	99.92%	99.93%	99.93%	99.95%

improvement in the solution for a predefined number of iterations.

4.6 Summary

In this chapter two separate investigations were carried out into the problem of resequencing trains through a junction in dynamic delay scenarios. Both investigations use a simulator that models a section of track on the Derby to Birmingham line taking in the North Stafford and Stenson Junctions. In both cases the problem is made dynamic by the addition of more trains over time as the original trains are still waiting to pass through the junction. In the first investigation only resequencing at the junction was considered. In this case the performance of P-ACO on several dynamic delay scenarios was compared to the performance of FCFS. Results showed that P-ACO outperformed FCFS when the changes were of a high to medium magnitude and a high to medium frequency.

In the second investigation the simulator was extended to also take into account trains sitting at the feeder stations to the junction. This is considered to be a first step towards expanding the problem to consider a larger area of the railway network. In addition, in the extended DRJRP, multiple delays were introduced in each change period and several different ACO algorithms were applied to the problem. Five ACO algorithms were implemented. Three of the algorithms were based on the P-ACO algorithm but with the introduction of immigrants to replace the memory after a

change. The fourth algorithm was based on MMAS, which has no inbuilt mechanism for adapting to change apart from the evaporation of pheromones trails. The fifth algorithm was a P-ACO algorithm that did not have the ability to reschedule the trains at the stations but simply dealt with them in the order that they arrived at the junction. In addition, a FCFS heuristic was used to investigate how the trains might be sequenced in a real-world situation.

Results showed that, on this dynamic rescheduling problem, FCFS was outperformed by all the ACO algorithms apart from on the high frequency changes where it has an advantage over RI-PACO and MMAS. In addition, ACO algorithms with a memory (P-ACO) were found to cope with dynamic changes better than an ACO algorithm that uses only pheromone evaporation (MMAS) to remove redundant pheromone trails.

It is apparent from the results that replacing the memory with elite immigrants in the extended DJRJP is an effective solution when the ants in memory cannot be modified to make them feasible in the new environment. In addition, in this dynamic problem, random immigrants were found to be unsuitable to replace the ants in memory when changes were of a high magnitude and a high frequency. The larger search space appears to demand the knowledge carried over from previous environments.

Surprisingly, adding the ability to sequence trains at the stations was not beneficial to this set of dynamic problems. This may be because the extra decisions that the ants have to make increases the size of the search space and the ants struggle to explore it adequately. This suggestion is supported by the fact that when the search space is relatively small, in the low change frequency scenarios, algorithms that sequence the trains at the stations (EI-PACO, HI-PACO and MMAS) often perform better than algorithms without station sequencing (NSS-PACO).

The next chapter introduces an extension to the DRJRP where a second objective is added to the problem to make it a multi-objective problem. The additional objective is that of minimizing energy consumption. As many real-world scheduling problems are multi-objective as well as dynamic, this is seen as an important step in the investigation into the application of ACO to real-world train rescheduling problems.

Chapter 5

Multi-objective Railway Junction Rescheduling in Dynamic Environments

Many railway rescheduling problems are not only dynamic but also multi-objective. The multi-objective nature of the problem is a result of the multiple demands placed upon the train dispatcher attempting to solve a rescheduling problem. They may need to simultaneously minimise several conflicting consequences of the perturbation, such as delay, timetable deviation, energy consumption and missed connections. The conflicting nature of these objectives means that increasing the quality of one objective might have a detrimental effect on the quality of another.

The aim of this work is to investigate the application of ant colony optimisation (ACO) to a difficult dynamic multi-objective optimisation problem (DMOP); the dynamic multi-objective railway junction rescheduling problem (DM-RJRP).

In the previous chapter, P-ACO was shown to be effective in solving a dynamic junction rescheduling problem. ACO also has the ability to cope with multi-objective problems due to its flexibility in being able to add multiple colonies, or multiple pheromone and heuristic matrices, to address the separate objectives. This suggests that ACO may be effectively applied to a dynamic, multi-objective railway rescheduling problem.

An advantage of using ACO for multi-objective problems is that its population-based nature means that multiple trade-off solutions can be generated in one run of the algorithm, in contrast classical optimisation methods may have to run the algorithm separately for each objective [29].

The unique contributions of this work are:

- The creation of benchmark problem and simulator to investigate dynamic

multi-objective railway rescheduling problems.

- The investigation of a railway rescheduling problem that is both dynamic and multi-objective. As previous works consider only static or multi-objective problems, they fail to take into account the dynamic and multi-objective nature of railway scheduling problems. The investigation of such a problem is a new contribution to railway rescheduling.

- A contribution to the field of understanding of how ACO algorithms can be applied to a railway rescheduling DMOP in terms of the features of the algorithms necessary for good performance and the effect of the frequency and magnitude of change on the algorithm's performance.

Several different multi-objective ACO (MOACO) algorithms are applied to the problem; based on a population based ACO (P-ACO) [94], and on the *MAX-MIN* Ant System (MMAS) [37]. Each algorithm uses a different method of dealing with dynamic changes. It is hoped that the performance of the algorithms will give insights into the features of the algorithm necessary for good performance on this DMOP. In addition, the best ACO algorithm is compared with NSGA-II [119], a 'state-of-the-art' multi-objective algorithm, and FCFS, a heuristic often used by railway dispatchers to resolve perturbations [32].

The problem has been modelled by extending the simulator created for the work in Chapter 4. The simulator evaluates the solutions produced by the algorithms in terms of two objectives: minimising timetable deviation and minimising additional energy expenditure. Minimising timetable deviation involves minimising the difference between a train's scheduled arrival time and its rescheduled arrival time, it attempts to ensure trains arrive neither too late or too early. The second objective minimises the extra energy consumed by the train as a result of changing the order that it passes through the junction. Energy usage is becoming more important to the successful performance of the the railway system [120].

This work has been accepted by IEEE Transactions on Intelligent Transportation as a regular paper [121].

5.1 The Dynamic Multi-objective Railway Junction Rescheduling Problem (DM-RJRP)

The DM-RJRP is concerned with the sequencing of trains through two junctions on the Derby to Birmingham line. It is a microscopic model as it is modelled at the level

of track block sections. The original static problem was created by Fan *et al.* [2] and has been extended in this paper to make it both dynamic and multi-objective.

As in Chapter 4, the two junctions under consideration are the North Stafford and Stenson Junctions on the Derby to Birmingham line. The problem is a dynamic one because as trains are waiting to be rescheduled at the junction more timetabled trains will be arriving, which will change the nature of the problem over time. A description of the dynamic nature of the problem can be found in Section 4.1.1.

5.1.1 The Problem Objectives

The DM-RJRP is not only a dynamic problem but also a multi-objective one. The two objectives under consideration are to minimise deviation from the original schedule and to minimise additional energy costs incurred by changing the order that the trains pass through the junction. These two objectives are described in more detail below.

Objective 1 - Minimising Timetable Deviation

Minimising timetable deviation involves minimising the difference between the train's new arrival time and its timetabled arrival time, whether that difference is positive or negative. In a rescheduling situation, trains that arrive too early can create as many problems as trains that arrive too late as both situations may result in conflict with other trains scheduled to use the same resources. The timetable deviation Dev_i of train i is calculated as in Eq. (5.1), where t_s is the scheduled arrival time and t_a is the actual arrival time.

$$Dev_i = |t_s - t_a| \tag{5.1}$$

The objective is to minimize the deviation, in minutes, for all trains at the point of change c, as shown in Eq. (5.2), where NT is the number of trains in the problem at change c.

$$f_1(c) = min \sum_{i=1}^{NT} Dev_i(c) \tag{5.2}$$

Objective 2 - Minimising Additional Energy Expenditure

The second objective is to minimise any extra energy expended by the trains as a result of changing the order that they pass through the junction. The energy usage calculation formulas were kindly supplied by associates at the University of Birmingham, and were taken from their microscopic railway simulator, BRaVE.

The energy expended by each train on its journey from its approach station to its destination station is calculated as follows.

$$Fg = resistance + \frac{wt * gt * gd}{cos(gd)} \tag{5.3}$$

$$F = Fg + wt * \frac{v - u}{\Delta t} \tag{5.4}$$

$$E = F * d \tag{5.5}$$

Eq. (5.3) calculates the force required to overcome gravity (Fg), where wt is the weight in kilograms of the train, gt is gravity (a constant value of 9.806) and gd is gradient (zero in this case as the track is level). The resistance for the train at its current speed is found using the look-up table provided by [108]. Eq. (5.4) calculates the force required to move the train (F), where v is the speed (in metres per second) the train is travelling at the end of the time step and u is the speed the train was moving at the end of the previous time step. Δt is the time step which is set to 1 second. Eq. (5.5) calculates the energy expended (E), where (d) is the distance travelled, in metres, in the current time step. The resulting value is in joules, it is converted to kWh by dividing by 3,600,000.

The objective is to mimimise the additional energy for all trains at the point of change c, as shown in Eqs. (5.6) and (5.7), where ExE_i is the additional energy expended by train i, E_s is the scheduled energy for train i, and E_a is the actual energy expended by train i.

$$ExE_i = argmax\{(E_s - E_a), 0.0\} \tag{5.6}$$

$$f_2(c) = min\sum_{i=1}^{NT} ExE_i(c) \tag{5.7}$$

The relationship between energy usage and train delay is complex. The original assumption made was that a slightly delayed train would use more additional energy than a seriously delayed train because the train would have had to travel faster to reduce the delay and the extra speed would use more energy. However, this was not found to be the case. Instead, a seriously delayed train was often found to use less energy. This is because, in the above equations, the amount of time a train spends waiting for the way ahead to clear before it can move again has no effect on the energy it consumes. When a train is waiting, d in Eq. (5.5) will be zero and consequently E will also be zero. It is recognised that, in the real world, a waiting train will use some energy, however, in the energy model represented by this set of energy equations that energy is not taken into account. A train that spends a lot of

time speeding up and slowing down to avoid conflict with other trains will expend more energy than a waiting train because acceleration, especially from a standing start, uses more energy than simply travelling at a constant speed. A seriously delayed train may have spent a larger proportion of its time waiting and less time speeding up and slowing down, therefore, it will have used less energy.

Preliminary experiments found that minimising energy usage and minimising timetable deviation do conflict to some degree. The smaller the difference between the trains scheduled arrival time and its original arrival time, the more energy it is likely to expend narrowing that difference. This may be because a train that arrives very near its original scheduled time will have spent very little time waiting but will have instead performed multiple slow-downs and speed-ups, in quick succession, to maintain its schedule. The multiple speed-ups mean that it will have expended more energy on its journey. As a result of this observed conflict between these two objectives, deviation rather than delay was chosen as the objective to minimise in this study.

For simplicity, in this work, only the extra energy expended by the trains is taken into account. Future work may take into account energy saved as well as energy expended.

The aim of this problem is to find a sequence of trains to pass through the junction to minimise the objective values. As the objective values are to some degree conflicting, there will not be a single solution to the problem but a set of trade-off solutions in form of a Pareto optimal set.

A characteristic of this problem is that the goal is to eventually remove the deviation and extra energy consumption from the system and to return the network to normal operation. Therefore, the aim is to end up with a single solution with an objective value of zero in each objective. This is different to many benchmark dynamic optimisation problems, where the problem constantly changes over time without ever being resolved. A further interesting feature of this problem is that it is time-linked. The decision made by the dispatcher as to which solution to choose to sequence the trains through the junction affects the trains that are available to reschedule at the next dynamic change.

5.1.2 The Stenson Junction Train Simulator

The simulator created for the work in Chapter 4 was extended by the addition of the second objective of minimising energy consumption. In this case, to simplify this already very complex problem the stations feeding into the junction were not taken into account.

Table 5.1: The scheduled timetable and energy consumption for each train

Train Number	Train Type	Route	Scheduled Arrival	Energy Consumption (kWh)
1	Class 150	A to D	12:10	23.96
2	Class 220	D to A	12:11	107.89
3	Freight	B to C	12:15	426.64
4	Class 220	D to B	12:16	63.33
5	Freight	B to D	12:16	307.15
6	Class 150	D to B	12:20	43.02
7	Freight	A to C	12:23	569.17
8	Class 150	C to A	12:21	67.90
9	Class 220	C to A	12:27	147.96
10	Class 220	B to C	12:30	140.82
11	Freight	C to B	12:39	434.57
12	Class 150	A to D	12:35	60.10

Table 5.1 shows the type of trains used, their routes through the junction, their scheduled arrival times and their original energy consumption. The timetable was created by running all trains, in their numerical order, through the simulator and recording their arrival times. This gave a baseline measurement to be able to calculate the deviation of the trains from their original timetable after a perturbation. As in Chapter 4, the trains are one of three types; Class 200, Class 150 or a F2-mixed freight train. Details of the length and maximum speed of each of these trains can be found in Section 4.1.3.

The simulator was made dynamic by the introduction of a specified number of trains (m) at specified time intervals (f). m represents the magnitude of change, while f relates to the frequency of change. The new trains were chosen by repeating the timetable shown in Table 5.1 in blocks of m trains.

The extra trains can be thought of as an extended timetable for the train junction and each combination of the magnitude and frequency of change was run through the simulator in order to obtain the conflict-free timetable for that dynamic scenario. A newly arrived train is not allowed to leave the station until the track section after the station is clear of all other trains. When a dynamic change occurs, any trains that have moved into the junction, or are about to move into the junction, are removed by the simulator and the remaining trains plus the additional trains are passed to the algorithm in timetable order.

5.1.3 Model Realism

At this present time, Network Rail are unable to provide the data necessary to investigate dynamic multi-objective railway rescheduling problems. Therefore, as a first step, a simulation tool has been developed that allows the investigation of such problems. To make the model as realistic as possible, it has been based on a real section of the British railway network, and the mechanics of railway operation, such as interlocking and automatic fixed block technology have been simulated. In addition the simulator uses the power and resistance data taken from RailSys data, which is used by Network Rail as a simulation tool [109]. The model allows the creation of delay scenarios with different combinations of magnitude and frequency of change. This facilitates investigation into the effect the characteristics of a dynamic change has on the ability of the algorithm to provide solutions.

5.1.4 Model Limitations

As in Chapter 4, the model is limited by the fact that it considers only a small area of the British railway network. In this multi-objective problem, where energy consumption is taken into account, a further limitation of the model is that it considers only flat sections of track with no gradients. The addition of gradients into the problem would affect the energy consumption of the trains and may impact the shape of the POF obtained by the algorithms.

5.2 Proposed MOACO Algorithms for the DM-RJRP

A description of the single-objective P-ACO and MMAS algorithms used in this work can be found in Section 2.2.2 and Section 2.2.3 respectively. In this section the changes made to adapt the algorithms to the DM-RJRP are described and discussed.

5.2.1 MOACOs for the DM-RJRP

There are many possible designs for MOACO algorithms as it is a popular research area and much work has been carried out on modifying ACO algorithms to make them suitable for multi-objective problems. In fact, work by López-Ibáñez and Stützle [91], where they automatically designed MOACOs for the symmetric bi-objective TSP, found that different designs produced similar quality results, suggesting that there is no single effective way to introduce a multi-objective aspect to

an ACO algorithm.

In this work, two multi-objective algorithms have been chosen for investigation. The first is a multi-objective version of P-ACO developed by Guntsch and Middendorf [94]. P-ACO is a population based ACO that has an inbuilt memory. This memory allows solutions from before the change to be carried over to the new environment thus retaining information already learned by the ants between changes.

The second algorithm used in this study is one designed by Alaya *et al.* [93] based on MMAS. This algorithm was chosen because its base algorithm was found to perform poorly on the DRJRP in previous work (see Chapter 4) and because its multi-objective modification, one colony and a pheromone matrix for each objective, is similar to that of multi-objective P-ACO. Choosing an algorithm that performed poorly makes it possible to investigate the modifications that are necessary to improve its performance on this DMOP. The following sections describe each of the algorithms in more detail with their dynamic adaptations.

5.2.2 Dynamic Multi-objective P-ACO

In the following sections, first the multi-objective version of the algorithm is described followed by the dynamic adaptation.

Multi-objective P-ACO

The single objective P-ACO algorithm [36] was adapted by Guntsch and Middendorf [94] to improve its performance on a multi-objective job shop scheduling problem where the two objectives to be minimised were overall tardiness and changeover costs. They modified the algorithm by adding a pheromone and heuristic matrix for each objective and by constructing the memory (P) from an archive of non-dominated solutions (Q). The memory is populated by choosing a random solution from Q plus $k-1$ closest solutions, where k is the size of the memory and the closeness of one solution to another is defined as the sum of absolute differences in objective values over all objectives. At the end of each iteration, any solutions that dominate the solutions in the archive are added to Q and any dominated solutions are removed from Q. The memory is populated from Q and the solutions in the memory are used to update each of the pheromone matrices. The ants' decision as to which node to choose next is based on a weighted summation of the separate matrices and the weights are determined using the average-weight rank method where the idea is to give an objective a higher weight if the solutions in P are better with respect to this objective compared to all solutions in Q.

In this work, two different versions of dynamic multi-objective P-ACO (DM-PACO) are compared. The first makes use of the average-weight-rank method proposed by Guntsch and Middendorf to facilitate the ants' decision making. This algorithm is referred to as DM-PACO-ST. The second version of the algorithm (denoted DM-PACO-R) randomly chooses which pheromone matrix to use at each decision point. This method is the same as that used by the multi-objective version of MMAS described in Section 5.2.3.

Dynamic Modification for P-ACO

This algorithm has an inbuilt memory and is able to retain information between changes. Therefore, the only modification introduced to this algorithm to allow it to cope with a dynamic change is to repair the solutions in the non-dominated archive. The repair involves removing any trains that have passed out of the problem and adding in any newly arrived trains in the order dictated by the train timetable. This is similar to the KeepElitist strategy used by Guntsch and Middendorf [24]. The pheromone values for any new trains added to the problem are initialised to τ_{min}. The repaired solutions are re-evaluated to assess their performance in the new environment and the members of the archive are reassessed for dominance: any solutions that are now dominated by any other solution in the archive are removed. The memory after a change is created from the non-dominated archive and used to reinitialise both pheromone matrices. The overall framework of this algorithm is given in Algorithm 1.

5.2.3 Dynamic Multi-objective MMAS

The multi-objective version of MMAS modified in this work is based on m-ACO$_4$(1,m), one of four algorithms designed by Alaya *et al.* for a multi-objective knapsack problem [93]. M-ACO$_4$(1,m) is similar to P-ACO in that it uses one ant colony with multiple pheromone structures, one for each objective. The following sections describe m-ACO$_4$(1,m) followed by details of the modifications made to attempt to improve its performance in a dynamic environment.

Multi-objective MMAS

In m-ACO$_4$(1,m), ants make their decision as to which node to choose next by randomly selecting one of the objective pheromone matrices to use in Eq. (2.1). At the end of an iteration, each pheromone matrix is updated separately for each objective using the best iteration ant for that objective. The update value Δ_x is

Algorithm 1 DM-PACO

```
1: Input P                                              ▷ The memory
2: Input Q                                              ▷ The non-dominated archive
3: Input k                                              ▷ The size of P
4: ConstructGraph
5: InitialisePheromoneTrails to τ_min
6: while (termination condition not satisfied) do
7:     ConstructSolutions
8:     EvaluateSolutions
9:     UpdateQ
10:    ClearP and update with k members of Q
11:    InitialisePheromoneTrails to τ_min
12:    UpdatePheromonesTrails                           ▷ using solutions in P
13:    if change occurs then
14:        ReconstructGraph
15:        InitialisePheromoneTrails to τ_min
16:        RepairSolutionsInQ
17:        EvaluateSolutionsInQ
18:        UpdateQ
19:        ClearP and update with k members of Q
20:        InitialisePheromoneTrails to τ_min
21:        UpdatePheromonesTrails                       ▷ using solutions in P
22:    end if
23: end while
```

based on the difference between the best-so-far ant's solution quality in objective x and the best iteration ant's solution quality in objective x as in Eq. (5.8), where S^x is the best solution in objective x for the current iteration and S^x_{best} is the best-so-far solution over all the iterations, including the current iteration, in objective x. The smaller the difference between the two, the larger the update.

$$\Delta_x = \frac{1}{1 + f_x S^x - f_x S^x_{best}} \tag{5.8}$$

As in the base MMAS algorithm, pheromone values are initialised to a maximum value. After each iteration, all pheromone trails are evaporated as in Eq. (5.9), where $L = E$ is the set of all pheromones and $0 < \rho \leq 1$ is the pheromone evaporation rate [22], which is a constant parameter of the algorithm. In addition, the pheromone trails are bound between a minimum value (τ_{min}) and a maximum value (τ_{max}).

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}, \ \forall (i, j) \in L, \tag{5.9}$$

Stagnation is addressed by reinitialising all trails to τ_{max} when the algorithm shows

Table 5.2: Four different versions of the DM-MMAS algorithm

	Clear Pheromones	Retain Pheromones
Clear Archive	DM-MMAS-SC	DM-MMAS-ST
Retain Archive	DM-MMAS-NC	DM-MMAS-NT

stagnation behaviour or there has been no change in the best fitness for a set number of iterations. To allow m-ACO$_4$ to produce a POS, it holds a non-dominated archive of solutions that it retains until the end of the run.

Dynamic Modifications for MMAS

MMAS has no inbuilt mechanism to cope with a dynamic change apart from the evaporation of pheromone trails, which can be slow [37]. The fact that MMAS was found to perform poorly on a single objective version of the DRJRP (see Sect 4.5.2) suggests that adaptations need to be made to m-ACO$_4$(1,m) to improve its performance on the multi-objective version of the DRJRP. The goal of the modifications is to investigate the role of the pheromone trails and the archive of non-dominated solutions in the algorithm's performance. Four different versions of the algorithm have been designed, summarised in Table 5.2, that either retain the pheromones and non-dominated archive after a change or clear them. The four designs are described in more detail below:

DM-MMAS-SC: The aim with this design is to investigate how important it is to retain the pheromones after a dynamic change. For this reason, the pheromone matrix is reinitialised to τ_{max} after the change to remove all the old pheromone information. In addition, the non-dominated archive is emptied of all solutions.

DM-MMAS-ST: This version is the closest to the original behaviour of MMAS after a change in Chapter 4. In this case, the pheromone values are retained after a change and only evaporation is used to remove old outdated decisions. As before, the non-dominated archive is emptied of all solutions.

DM-MMAS-NC: Here the aim is to investigate the importance of retaining the non-dominated archive between changes. Therefore, the non-dominated archive of solutions is retained after a dynamic change. However, as the archive is no longer relevant to the new environment, the solutions in it have to undergo a repair. The same repair strategy is used as for DM-PACO (see Section 5.2.2),

Algorithm 2 DM-MMAS-SC

```
1: Input NDS                                          ▷ The non-dominated archive
2: Input r                                            ▷ Reinitialisation Interval
3: Input BestIterationAnt_i                           ▷ in objective (i)
4: ConstructGraph
5: InitialisePheromoneTrails to τ_max
6: while (termination condition not satisfied) do
7:     ConstructSolutions
8:     EvaluateSolutions
9:     UpdateNDS                    ▷ with any new non-dominated solutions
10:    Update BestIterationAnt_i
11:    EvaporatePheromoneTrails
12:    UpdatePheromone i                          ▷ using BestIterationAnt_i
13:    if no change in BestIterationAnt_i for r iterations then
14:        ReinitialisePheromone i to τ_max
15:    end if
16:    if change occurs then
17:        ReconstructGraph
18:        InitialisePheromoneTrails to τ_max
19:        EmptyNDS
20:    end if
21: end while
```

but, in this case the pheromone values for the new trains added to the problem are initialised to τ_{max}. In addition, the pheromone trails are cleared after each change.

DM-MMAS-NT: The purpose of this modification is to investigate both the importance of the pheromone information and the non-dominated archive after a dynamic change. Therefore, both the non-dominated archive and the pheromone information are retained after a change.

The framework of the base DM-MMAS algorithm without modifications (DM-MMAS-SC) is given in Algorithm 2.

5.2.4 Dynamics Implementation

Solving a real-world train rescheduling problem requires consideration of how it could be implemented in a real-world railway perturbation scenario. After a delay, the trains relevant to the problem are passed to the algorithm to discover a POS of solutions. The train dispatcher then chooses the solution that best matches their objectives at that moment in time. The sequence of trains in this solution is run through the junctions until a dynamic change occurs, triggered by the arrival of more

timetabled trains. At the point of change, a 'snapshot' is taken of the junctions by the simulator. The snapshot records the status of the trains, track and junction at the point of change. The newly arrived trains and the snapshot are passed to the ACO algorithm and the algorithm is run again to find a POS of solutions for the new environment. The first action the algorithm takes when it receives the updated information is to reconstruct the directed edge graph that the ants walk along to make their solutions. This is necessary since some trains will have passed through the junction and will no longer be relevant to the problem while other trains will have been added.

As it is not possible to predict the solution that a train dispatcher might select from the set of non-dominated solutions presented to them, this choice is simulated within the algorithm. This is achieved by randomly choosing a solution from the POS to make the snapshot of the junction at the point of change.

5.2.5 Comparison Algorithms

To compare the ACO algorithms with other approaches for the same problem, the experiments were repeated using NSGA-II [119], a 'state-of-the-art' multi-objective algorithm. NSGA-II is traditionally applied to continuous optimisation problems. In order to allow it to be used for this combinatorial problem, where the order of trains that pass through the junction has to be feasible, the crossover and mutation operators were modified. The purpose of crossover is to exploit previously found good solutions, Therefore, it was replaced with an operation that, with probability p_c, performs a path-preserving local search with a parent to create a new child solution. Details of this search procedure can be found in Section 4.2.2. The purpose of mutation is to explore the search space. Therefore, it was replaced with a procedure that, with probability p_m, replaces a parent with a random feasible child solution.

In addition, the performance of the proposed MOACO algorithms is compared with FCFS. The comparison is limited by the fact that FCFS can produce only a single solution to the problem. However, this approach is included to allow comparison with a technique used in the railway industry [32].

5.3 Experimental Study

5.3.1 Experimental Design

Algorithm parameters were established by preliminary experimentation. The best combination for DM-PACO was found to be 12 ants with a memory size of 8. Such

a large memory size means that in many cases all the ants in the non-dominated set will be included in the memory and therefore the memory completely reflects the non-dominated set. A q_0 value of 0.0 was found to perform best, which results in the ants always making a probabilistic decision about the next node to choose. For both pheromone matrices, the maximum pheromone value (τ_{max}) was set to 1, the minimum pheromone value (τ_{init}) was set to $1/n$, where n is the number of nodes, and the pheromone update value to $(\tau_{max} - \tau_{init})/k$, where k is the size of the memory. All pheromone levels were initialised to τ_{init}.

To make the algorithms more comparable, the same number of ants were used for each algorithm. The pheromone bounds for MMAS are given by $\tau_{max} = \frac{1}{S}$ and $\tau_{min} = \frac{\tau_{max}}{a}$, where S is the fitness of the best ant and a is a constant parameter of the algorithm. For both pheromone matrices, a was set to 25 and p to 0.5. As in the original MMAS algorithm [22], the q_0 value was set to 0.0. Reinitialisation of the pheromone matrices to maximum was triggered when there had been no change in the best-so-far solution after 20 iterations.

In the case of NSGA-II, preliminary experimentation showed that the best performance was obtained with a p_c of 0.2 and a p_m of 0.2. To make it comparable with the ACO algorithms, a population size of 12 was used.

Nine different dynamic environments were investigated involving all permutations of 3 different magnitudes of change (8 trains, 5 trains, 2 trains) and 3 different frequencies of changes (5 mins, 10 mins, 15 mins). For all algorithms, the POS at the point of change was recorded. Thirty runs were completed for each algorithm on each dynamic scenario and all algorithms were run for 125 iterations before a dynamic change.

5.3.2 Performance Measures

The two goals of multi-objective optimisation are to find solutions that are as close to the POF as possible and as well spread as possible along the whole POF [29]. However, this is a difficult task in the real world when the true POF is unknown. For this reason, two different performance measures have been adopted to give insight into the performance of the algorithms. The measures are hypervolume (HV), which measures how much of the objective space is dominated by the members of the POS [87], and generational distance (GD), which measures the convergence of a solution's non-dominated set towards the POF. To calculate HV, a reference point is required. In this study, it is determined by the worst values for each objective over all algorithms and over all changes. This is to allow the values across all changes to be averaged as they all use the same reference point. To calculate GD, a reference

Table 5.3: A statistical analysis of average HV at 0.05 significance level

	$m = 8$			$m = 5$			$m = 2$		
$f =$	5	10	15	5	10	15	5	10	15
DM-MMAS-NT ⇔ DM-MMAS-ST	$s+$	$s+$	$\sim$	$s+$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
DM-MMAS-NT ⇔ DM-MMAS-NC	$s+$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
DM-MMAS-ST ⇔ DM-MMAS-SC	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
DM-PACO-R ⇔ DM-MMAS-NT	$\sim$	$s+$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
DM-PACO-R ⇔ NSGA-II	$s+$	$s+$	$\sim$	$s+$	$s+$	$\sim$	$\sim$	$\sim$	$\sim$
DM-PACO-R ⇔ FCFS	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$

POF (POF^R) is needed. As this is a real-world problem, the POF^R is unknown and therefore is created for each delay scenario from the union of all the POS for all the algorithms for that particular change. To give an offline performance measure for each run (P_r) an average was taken over all the changes, see Eq. (5.10), where NC is the number of changes and PM is the performance measure under consideration, either HV or GD.

$$P_r = \frac{1}{NC} \sum_{c=1}^{NC} PM_c \tag{5.10}$$

5.3.3 Experimental Results

The first interesting result is that there is no significant difference between DM-PACO-ST and DM-PACO-R across all the scenarios on both of the performance measures. In DM-PACO-ST, the ants base their decisions on a weighted aggregation of the pheromones for each objective using weights determined by the average-weight rank method [94], whereas in DM-PACO-R, the ants make their decision using only one, randomly selected, pheromone matrix. As there is no difference between the two versions of DM-PACO, DM-PACO-R was chosen as the comparison algorithm as it uses the same decision method as that used by DM-MMAS.

Results were tested for statistical significance using the Kruskal-Wallis test for multiple comparisons followed by the Wilcoxon rank-sum pairwise test with Bonferroni correction at a 0.05 significance level. FCFS was compared to DM-PACO-R using the one-sample Wilcoxon signed rank test as in this case a single FCFS result is compared with multiple results for the ACO algorithm. Table 5.3 relates to the HV performance measure. Results for the GD performance measure were similar. The table shows the results of comparing Algorithm1 ⇔ Algorithm2, where the symbol "$s+$", "$s-$" or "$\sim$ " indicates that Algorithm1 is significantly better than,

significantly worse than, or not significantly different from Algorithm2, respectively.

For both HV and GD, the version of m-ACO$_4$(1,m) that retains the non-dominated set and the pheromone trails after a change (DM-MMAS-NT) performs significantly better than the version that retains the pheromone trails but clears the non-dominated set after a change (DM-MMAS-ST). This significant difference in performance is apparent on the high magnitude, high and medium frequency changes ($m = 8, f = 5$ and $m = 8, f = 10$) and the medium magnitude, high frequency change ($m = 5, f = 5$). This result suggests that, in the DM-RJRP, it is very important to retain the non-dominated archive of solutions between changes when the changes are of a high frequency and of a medium to high magnitude.

Retaining the non-dominated archive between changes can be thought of as keeping a memory of the solutions found before. The continued existence of this archive provides a set of solutions to compare any new solutions to, when checking for dominance. When many trains are added in short intervals, few trains will have had the opportunity to pass through the junction before the next set of trains arrives. This results in a large number of trains in the system and a correspondingly large search space for the ants to navigate. The large search space may make it difficult for the ants to find good new solutions especially as the good solutions may now have become localised in one area of the search space due to time-linked nature of the problem. Retaining and repairing the archive means that only solutions that are better than those already found are added to the archive, which guides the algorithm in its search for better solutions.

With regards to the issue of retaining pheromone values between changes, a comparison between DM-MMAS-NT and DM-MMAS-NC shows that, on the high magnitude, high frequency change ($m = 8, f = 5$), when both the pheromone trails and the non-dominated archive are retained between changes, the algorithm performs significantly better than when the non-dominated archive is retained but the pheromone trails are cleared. These results suggest that, in the high magnitude high frequency change, retaining the pheromones between changes improves the performance of the algorithm.

However, DM-PACO-R still significantly outperforms DM-MMAS-NT on the high magnitude, medium frequency change ($m = 8, f = 10$) even though they both retain the non-dominated archive between changes. This suggests that there may be more improvements needed for DM-MMAS to allow it to perform well in this DMOP. It is interesting that this performance difference is seen for the high magnitude, medium frequency scenario ($m = 8, f = 10$) rather than for the high magnitude, highest frequency scenario ($m = 8, f = 5$). In the previous work in

Table 5.4: Number of times FCFS dominates the POS produced by DM-PACO-R in each delay scenario (square brackets denote the change number)

m=8			m=5			m=2		
f=5	f=10	f=15	f=5	f=10	f=15	f=5	f=10	f=15
0	0	0	0	0	0	1[7]	0	1[3]

Chapter 4, the high magnitude, high frequency scenario showed the biggest difference in performance between the algorithms. However, an examination of the underlying results suggests that scenario $m = 8, f = 10$ is, in this DMOP, more difficult to solve than $m = 8, f = 5$. In $m = 8, f = 5$, the DM-PACO-R algorithm converged to the desired solution (0, 0) in 50% of the runs, while in $m = 8, f = 10$ it only achieved convergence in 3.33% of the runs. This suggests that $m = 8, f = 10$ is the more difficult problem to solve and also suggests that, in this DMOP, the difficulty of the problem is not only determined by the magnitude and frequency of dynamic change but also by the interaction between the objectives.

Table 5.3 shows that NSGA-II performs as well as DM-PACO-R on all the low magnitude changes ($m = 2$) and on the high and medium magnitude low frequency changes ($m = 8$, $f = 15$ and $m = 5$, $f = 15$). This suggests that the crossover and mutation operators used are a viable answer to the problem of how to preserve a workable order of trains to pass through the junction. However, DM-PACO-R significantly outperforms NSGA-II on the high and medium magnitude and high and medium frequency changes. This is most likely because NSGA-II has no inbuilt mechanism to cope with dynamic change and also does not retain its non-dominated archive between changes. This provides further evidence for the importance of retaining the non-dominated archive between changes.

FCFS is outperformed across all scenarios by DM-PACO-R. This is not unsurprising as FCFS produces only a single solution which may result in a lower HV score than a set of 'trade-off' solutions. For this reason, the number of times the single solution produced by FCFS dominates the solutions in the POS produced by DM-PACO-R was examined (Table 5.4). In this table, the value in square brackets shows the change number where the solution in FCFS dominated the solutions in DM-PACO-R. The table shows that FCFS only dominates the solutions in DM-PACO-R when $m = 2, f = 5$ and when $m = 2, f = 15$. In each case, it was for a single change, change 7 for $m = 2, f = 5$ and change 3 for $m = 2, f = 15$. This result shows that for all the high/medium magnitude and high/medium frequency changes FCFS produces solutions that are worse than the solutions in the POS for DM-PACO-R. This is illustrated in Fig. 5.1, where it can be seen that the single

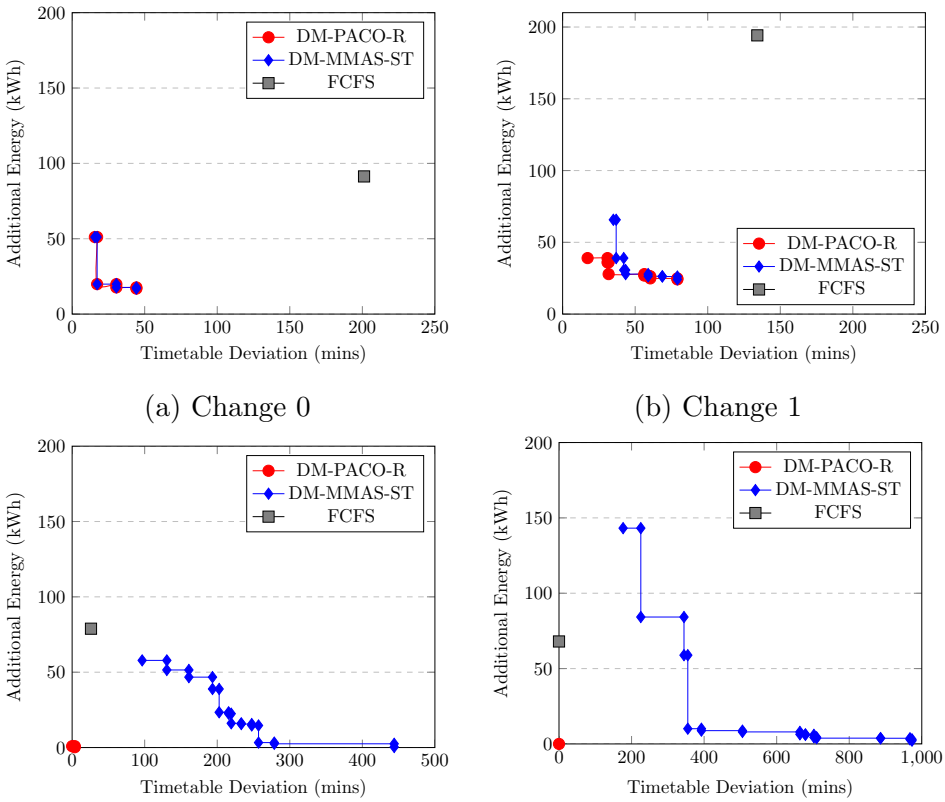


Figure 5.1: Amalgamated POFs for DM-PACO-R and DM-MMAS-ST on m=8 f=10 for a selection of change instances.

FCFS solution is worse than the POF produced by DM-PACO-R.

Fig. 5.1 shows the evolution of the POF over time for the best performing algorithm (DM-PACO-R) and one of the worst performing algorithms (DM-MMAS-ST) for the delay scenario involving 8 additional trains introduced every 10 minutes. They show the POFs produced when a non-dominated set is created from the union of all the runs. The points on the front are not joined to give a smooth representation as this may be misleading for two reasons: 1). there is no guarantee that the front actually is smooth and 2). actual solutions corresponding to the intermediate vectors are unknown and may not actually exist [122]. The scale of each graph varies to make it easier to see the POFs produced.

It is apparent from Fig. 5.1 that the algorithms can solve the DM-RJRP to produce a POS of trade-off solutions. It is also apparent that the non-dominated fronts produced are very different for the two algorithms. Before any additional trains have been added to the problem both algorithms find a very similar POF, it is only after more trains are added that the shapes of the fronts start to diverge.

Table 5.5: Example trade-off solutions for change 1 $m = 8$ and $f = 10$ for DM-PACO-R for each member of the best-so-far POS

Deviation (min)	Add. Energy (kWh)	Train Order
17.217	39.014	6-8-9-7-1-11-10-12-13-14-15-16-17-18-19-20
43.550	27.905	6-8-9-7-11-10-14-16-1-12-13-15-17-18-19-20
58.850	26.839	6-8-9-7-11-10-1-14-15-16-17-18-20-12-13-19
75.750	26.362	6-8-9-7-11-10-14-16-1-20-18-15-17-12-13-19
85.450	24.173	6-8-9-11-10-14-7-1-15-16-18-17-12-13-19-20

Table 5.6: FCFS solution for change 1 $m = 8$ and $f = 10$

Deviation (min)	Add. Energy (kWh)	Train Order
134.167	194.201	8-12-6-3-9-5-14-10-16-13-11-15-18-17-19-20

After change 4, we can see a dramatic difference in the two fronts, with DM-MMAS-ST producing a large front with many solutions while DM-PACO-R has converged to a single solution with a value of zero in each objective. At a first glance, the set of graphs for DM-MMAS-ST looks the most promising as it shows a large number of non-dominated solutions on the POFs. However, paradoxically, this is not what we want in this real-world problem. In contrast, we want the effects of the delay to eventually disappear from the system to allow the trains to return to their normal running schedule. DM-PACO manages to achieve this.

It is interesting to note that overall the number of non-dominated solutions produced in this real-world problem is actually very small. This is similar to results obtained by Corman *et al.* in their work on bi-objective conflict resolution in railway traffic management [16]. In two out of three of their scenarios, they obtained an average of only 3 or 4 Pareto optimal solutions. This suggests that small numbers of Pareto optimal solutions may be a feature of real-world railway rescheduling problems.

Tables 5.5 and 5.6 show an example set of non-dominated solutions produced for DM-PACO-R and for FCFS. Each row in Table 5.5 is a non-dominated solution with the deviation and additional energy incurred. The train order is the order the trains need to pass through the junction to give those values. The tables show that FCFS has a different set of trains to sequence than DM-PACO-R. This is because, before the change occurred, different trains were sequenced and removed from the problem by FCFS than by DM-PACO-R thus resulting in a different set of trains for

Table 5.7: Average algorithm execution times in **minutes** for DM-PACO-R on scenario $m = 2,\ f = 15$ and $m = 8,\ f = 5$

Change	1	2	3	4	12
$m = 2, f = 15$	0.40	0.22	0.21	0.25	-
$m = 8, f = 5$	1.23	1.82	2.49	3.20	11.50

each algorithm to work with. This illustrates the time-linked nature of the problem.

5.3.4 Algorithm Computation Times

The experiments were run on a 2.9GHz Intel Xeon E5-2666 v3 (Haswell) processor. Table 5.7 shows the average execution times for dynamic scenarios $m = 2, f = 15$ and $m = 8, f = 5$. The times are shorter than those in Chapter 4, Section 4.5.3 because the experiments were run on a faster processor. These two scenarios were chosen as they are the extremes of the delay scenarios. The timing results for all algorithms were similar, therefore only the results for DM-PACO-R are shown.

Over all changes, when two trains are added every 15 m ($m = 2, f = 15$), the average execution time is less than a minute. However, when eight trains are added every five minutes ($m = 8, f = 5$), the large number of trains in the problem increases the work of the simulator and results in an average execution time of 11.50 m for change 12. This large computation time is, of course, unacceptable in a real-world situation. However, it could be reduced by choosing a different termination condition, e.g., running the algorithm until there has been no improvement in the solutions for a predefined number of iterations. In addition, ACO is very amenable to being run in parallel [22], which would cut down the computation time considerably and make it feasible for real-time operation.

5.4 Summary

In this chapter, the dynamic Stenson junction simulator from the previous chapter has been extended to introduce a second objective, that of minimising additional energy consumption. This has enabled an investigation into the application of ACO algorithms to a dynamic multi-objective railway rescheduling problem. The investigation of DMOPs in the railway industry is a little explored area, as is the application of ACO algorithms to such problems.

An additional goal of this work was to attempt to identify the features of an ACO algorithm that make it suitable for coping with both the dynamic as well as

multi-objective nature of this problem. The study involved the use of several multi-objective ACO algorithms. Two of the algorithms were based on multi-objective P-ACO, the others were based on different variations of a multi-objective MMAS algorithm where the aim of the modifications was to improve the performance of the algorithm on the DM-RJRP.

It is apparent that all the ACO algorithms can find a POS of solutions for the DM-RJRP. However, the algorithm based on P-ACO performs better than the algorithms based on MMAS. The performance of multi-objective MMAS can be improved, on this problem, by retaining the non-dominated archive between changes. However, for a comparable performance with DM-PACO-R, on scenarios with large and frequent changes, multi-objective MMAS also benefits from retaining the pheromone trails between changes. The best performing algorithm DM-PACO-R also outperformed NSGA-II and FCFS.

An interesting observation in this work is that a scenario that was more difficult for the algorithm to solve in the dynamic single objective version of this problem was not necessarily the most difficult scenario to solve when the problem was made multi-objective. This suggests that the problem difficulty is not only influenced by the magnitude and frequency of dynamic change but also by the interaction between the objectives.

This work has concentrated on modifications to the algorithm after it encounters a dynamic change. It is feasible that the internal mechanisms of the algorithms may also have an effect on their ability to solve this DMOP. For example, DM-MMAS updates the pheromones with the best iteration ant in each objective while DM-PACO updates with the ants in a memory created from the non-dominated set. In addition, it is possible that NSGA-II's performance could be improved by modifications to make it able to retain information between changes such as the introduction of elite immigrants. In future work, the aim is to investigate the effect of these internal mechanisms on the algorithms' performance.

The fact that the model used to explore this problem simulates the physical movement of trains through the junctions, means that on the high magnitude, high frequency changes the time taken to produce a solution is unrealistically long. In addition, this work is focused on a small area of the railway network and does not take into account the effect that changes made in a local area will have on the global behaviour of the network. For this reason, the railway model created for the next chapter is an event-based, macroscopic model of the railway that takes into account the movements of the trains between timing points on a train's journey. This new model allows the assessment of the impact local decisions, made at the station, have

on the trains' ongoing journeys.

Chapter 6

Station Rescheduling in Dynamic Environments

It is very apparent, when commuting from Leicester to Nottingham, that delays to trains arriving at Leicester station are a common occurrence. A delayed train will miss its scheduled time slot on the platform and will have to be reallocated to a new time slot either on its original platform or on a different platform. The reallocation may results in secondary delays to trains that may have to be delayed themselves to allow the delayed train to be accommodated. This raises the question of whether it is possible to use ACO to determine the best platform to allocate to a delayed train to minimise the overall delay in the system. However, it is not enough to make a local decision about the best platform to place the platform on as this does not take into account the ongoing journeys of the affected trains. Therefore, one of the aims of the work, in this chapter, is to not only investigate reallocating trains to new platforms but to take into account the effect the local reallocation decisions have on subsequent conflicts at the timing points on the remainder of each train's route.

The potential usefulness of this forecasting approach was illustrated by a discussion with the Station Master and his dispatch team at Birmingham New Street station. They revealed that often, when reallocating delayed trains to platforms, they will look ahead to the rest of the trains journey to see the impact the local rescheduling decision will have. If part of this process could be automated it would allow the dispatcher to consider the ongoing effect of many more rescheduling options in the short time they have available to reschedule delayed trains.

The motivation behind this work was the desire to demonstrate the applicability of ACO algorithms to a real-world problem and to provide a step forward in the development of a real-time dispatching system that could be used to help decide which platform to allocate to a delayed train to minimise delay. To make the problem

as realistic as possible a model of the station was created using Network Rail's schedule data from the Integrated Train Planning System (ITPS) [30]. The model details both the movement of trains through the station and the movement of all trains at each of the timing points on the trains' routes. This allows the long-term consequences of the reallocation decisions to be determined. The use of real schedule data makes the results applicable to a real-world dispatching system as the same data is available to the dispatchers when rescheduling trains.

Real world train delay data is not available for the British railway network at the current time. Network Rail produce a train movements data feed which could be used to monitor delay but as there is no information about the cause of the delay it would be unfair to compare the results of the real delay resolution with the solution found by the algorithm. For example, if the real delay was caused by a track blockage, the ACO algorithm would have no knowledge of this and may find an improved solution because it included the blocked track section in its rescheduling decisions. Therefore, in order to investigate the effect of the magnitude and frequency of the delay on the performance of the algorithms, different magnitudes and frequencies of delays were simulated in the model.

The problem of reallocating trains to platforms after a delay can be divided into two sub-problems. The first problem is to decide the platform to allocate to the delayed train (the reallocation sub-problem); the second is to decide the order that the trains should leave the station (the resequencing sub-problem). Solving these sub-problems can allow trains to overtake other trains at the station. This is important because often the limitations of the railway infrastructure means that overtaking is only possible at stations [21]. In this work two colonies of ants have been used to address each of these sub-problems. Investigations have been carried out to examine the effectiveness of using each colony separately and of combining them into a multi-colony algorithm.

The unique contributions of this chapter are:

- The creation of a dynamic benchmark problem for a dynamic station rescheduling problem based on Network Rail's schedule feed.
- A contribution to the field of understanding of how ACO algorithms can be applied to the field of dynamic railway rescheduling, specifically dynamic platform reallocation and resequencing.
- The creation of a unique framework that combines two colonies of ants, one that reallocates trains to the platforms after a delay and the other that determines the order the trains leave the station.

In addition a novel best-ant-replacement scheme has been introduced in this work that takes into account not only the delay that results from a particular set of platform reallocations but also the physical distance between the train's original and its new platform. Results showed that this modification worked well to minimise platform displacement where platform displacement is a measure of the physical distance between a train's original platform and its reallocated platform. The smaller the platform displacement the shorter the travel distance from the train's original platform to the reallocated platform and the smaller the inconvenience to passengers and rail crew.

To evaluate the effectiveness of using ACO algorithms for this problem, their performance is compared with those obtained using TS, VNS and running with no platform reallocation or resequencing (NO-ALG). TS has previously been used by Corman *et al.* [54] for railway rescheduling while VNS has been applied by Samà *et al.* [10]. In both cases, the algorithms were used within the AGLIBRARY optimisation solver of ROMA and were found to improve performance.

Part of this work was previously published in [123].

6.1 The Dynamic Station Rescheduling Problem (DSRP)

6.1.1 Description of the Problem

The DSRP is the problem of reallocating multiple, successive, delayed trains to new time-slots at a railway station with the aim of minimising the ongoing delay in the system. The work in this chapter concentrates on reallocating trains at Leicester station. Leicester is a busy, bottleneck station with both passenger and freight trains coming from four different directions [124]. It contains four, bidirectional, passenger platforms with trains able to enter and leave any platform from any adjoining track section.

The effect of the platform reallocation decisions are not only considered locally, at the station, but also at all the timing points on the remainder of each train's journey within a specified radius of the station. In this way, it is possible to take into account both the immediate and future outcome of the platform reallocations. The radius considered is 50 miles (80.47 km) of the station which covers approximately 225 timing points. Fig. 6.1 shows some of the timing points within the station radius.

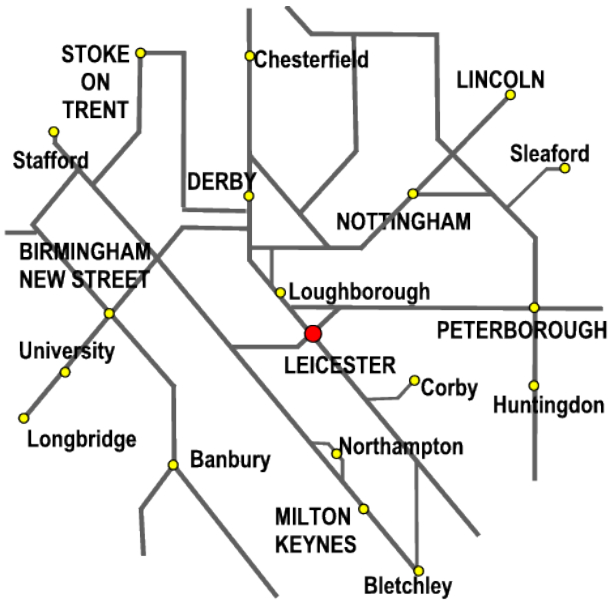


Figure 6.1: Railway Network Diagram of Timing Points in a 50 mile (80.47 km) radius around Leicester (not drawn to scale)

6.1.2 Mathematical Model for the DSRP

Each train (t) in T, the set of all trains, has a route that consists of a list of timing points and the times it arrives at and departs each point. For each timing point r, in the set of all timing points in the problem (TP), there is a list of events (E_r) with each event corresponding to the arrival and departure of a train at that timing point. A train may be associated with more than one event at a timing point if it makes a return journey.

A track section is a length of track between two timing points. A timing point may be associated with more than one track section. At non-station timing points there are assumed to be at least two track sections, that are traversed in opposite directions. However, at a station, there will be several track sections, each corresponding to a platform. B_r is the set of track sections for timing point r.

The problem involves three decision variables; P_k, the platform assigned to event k for train t at the station, x_k^{arrive} and x_k^{depart} the arrival and departing times of event k associated with train t at the station and at each of the timing points on its ongoing journey. Table 6.1 describes the relevant notations used in the mathematical model.

Table 6.1: Notations and descriptions for the mathematical model

Notation	Description
T	is the set of all trains in the evaluation window
t	is the index of a train in the evaluation window, $t \in T$
TP	is the set of all timing points in the problem
r	is the index of a timing point, $r \in TP$
B_r	is the set of all track sections, for one timing point r, $r \in TP$
b	is the index of a track section at a timing point, $b \in B_r$
E_r	is the set of all events at timing point r
E_s	is the set of all events at the station under investigation
k	is the index of an event in E_r, $k \in E_r$
$k+1$	is the next event after event k in E_r
$P_k^{original}$	is the original platform for event k, $k \in E_s$
P_k^{new}	is the new platform for event k, $k \in E_s$
P_k^{static}	is the forced platform for event k, $k \in E_s$
$a_{k,b}^{initial}$	is the initial scheduled arrival time of event k on track section b
$d_{k,b}^{initial}$	is the initial scheduled departure time of event k on track section b
$a_{k,b}^{static}$	is the forced arrival time of event k on track section b
$d_{k,b}^{static}$	is the forced departure time of event k on track section b
$x_{k,b}^{arrive}$	is the reassigned arrival time of event k on track section b
$x_{k,b}^{depart}$	is the reassigned departure time of event k on track section b
$(x_{k,b}, x_{k+1,b})$	is a pair of events that use the same physical train t, $t \in T$, where k is a terminating event and $k+1$ is an originating event, $k \in E_r$
$l_{k,t}$	is the last event k for train t, within the radius and the evaluation period, $k \in E_r$, $t \in T$
$l_{k,b}$	is the last event k for track section b, $k \in E_r$, $b \in B_r$
Z_t	represents the delay train t experiences when departing $l_{k,t}$

Table 6.1 (Continued)

Notation	Description
N	the number of station trains in the current evaluation window
C	is the set of all change periods
c	is the index of a change period, $c \in C$
sp_c	is the start time of the problem in change period c
ep_c	is the end time of the problem in change period c
$time_{not}$	is the notification interval, how far in advance the train controller receives notification of the train delay
$time_{plan}$	is the planning period, how far into the future the train controller decides to reshuffle trains on the station platforms

In Table 6.1 the forced platform (P_k^{static}), forced arrival time ($a_{k,b}^{static}$) and forced departure time ($d_{k,b}^{static}$) of event (k) refers to an event associated to a train that arrives before the start time of the problem and therefore its platform, departure and arrival time cannot be changed.

Soft Constraints

To reduce disruption to passengers' journeys it is desirable to reallocate a delayed train to a platform close to the original platform. Constraint (6.1) expresses this, $posP_k^{original}$ and $posP_k^{new}$ are the physical positions of the original and the new platforms respectively. It is a soft constraint because although desirable it is not essential to satisfy it for the safe operation of the railway.

$$min(posP_k^{original} - posP_k^{new}) \quad \forall k \in E_s \tag{6.1}$$

Hard Constraints

It is essential to satisfy hard constraints for the safe and efficient running of the railway.

$$x_{k,b}^{depart} < x_{k+1,b}^{arrive} \quad b \in B_r, k \in E_r : k \neq l_{k,b} \tag{6.2}$$

Constraint (6.2) ensures the end time of an event at timing point r on track section b is less than the start time of the next event on that track section. This determines that there is no overlap between trains occupying the same track section and thus that only one train can occupy a track section at one time.

$$x_{k,b}^{depart} \geq d_{k,b}^{initial} \quad b \in B_r, k \in E_r \tag{6.3}$$

Constraint (6.3) ensures that the new departure time of event k on track section b is not earlier than the original departure time specified in the train schedule. This constraint is especially important on the station platforms as trains that depart earlier than their scheduled departure time will cause chaos for passengers.

$$P_k^{new} = P_{k+1}^{new} \quad \forall \quad x \in (x_{k,b}, x_{k+1,b}) \tag{6.4}$$

Constraint (6.4) ensures the platform for a pair of events $(x_{k,b}, x_{k+1,b})$ that use the same physical train t, $t \in T$ remains the same for each event in the pair. That is event $x_{k,b}$ cannot be assigned a different platform to $x_{k+1,b}$. In this case $x_{k,b}$ is a terminating event, while $x_{k+1,b}$ is an originating event.

$$x_{k-1,b}^{depart} < (x_{k,b}, x_{k+1,b})^{arrive} \quad b \in B_r, k \in E_r : k \neq l_{k,b} \tag{6.5}$$

$$(x_{k,b}, x_{k+1,b})^{depart} < x_{k+2,b}^{arrive} \quad b \in B_r, k \in E_r : k \neq l_{k,b} \tag{6.6}$$

Constraints (6.5) and (6.6) ensure that trains cannot be placed on the platform between pairs of events that use the same physical train t, $t \in T$. They can be placed before or after the paired events but not between them.

The start time (sp_c) of the problem is the time that the station controller is notified of the delayed train as described in Eq. (6.11) where $time_{not}$ is how far in advance the train controller is notified of the train delay, for example 30 min.

$$sp_c = a_{k,b}^{initial} - time_{not} \tag{6.7}$$

Any trains that arrive before this time but depart after this time are transition trains that straddle the problem boundary. As they arrive before the start of the problem, they are not considered for reallocation to a new platform and their arrival time and departure time at the station is fixed. However, their arrival and departure at the timing points on their ongoing journey may be changed if they conflict with other trains wanting to use the same track section at the same time. Constraints (6.8), (6.9) and (6.10) ensure that the platform and arrival time for transition trains remain unchanged at the station.

$$P_k^{new} = P_k^{static} \; k \in E_s : x_{k,b}^{arrive} < sp, x_{k,b}^{depart} \geq sp \tag{6.8}$$

$$x_{k,b}^{arrive} = a_{k,b}^{static}\ k \in E_s : x_{k,b}^{arrive} < sp, x_{k,b}^{depart} \geq sp \tag{6.9}$$

$$x_{k,b}^{depart} = d_{k,b}^{static}\ k \in E_s : x_{k,b}^{arrive} < sp, x_{k,b}^{depart} \geq sp \tag{6.10}$$

Trains that arrive and depart at the station before the station controller is notified of the delay remain unchanged in terms of allocated platform and arrival and departure at the station. However, if any timing points on their ongoing journey fall within the evaluation period then the arrival and departure at those timing points may be changed to remove conflict with other trains.

The end time (ep_c) of the problem for each change, c, depends on how far into the future the station controller wishes to include trains for consideration in the station reallocation problem. Eq. (6.11) illustrates this, where $time_{plan}$ is the time period after the delayed arrival time of the train that the train controller will consider trains for reshuffling at the station. For example a station controller may decide that all trains that arrive up to 30 min after the new arrival time of the delayed train can be included in the reallocation problem. The larger the value of $time_{plan}$ the more trains will be included in the problem.

$$ep_c = x_{k,b}^{arrive} + time_{plan} \tag{6.11}$$

The evaluation period determines how far into the future the controller wishes to assess the impact of the decisions made at the station. This determines which timing points on the trains route are included in the problem.

6.1.3 The Objective

When a train is delayed, it will miss its scheduled time-slot and its new arrival time may create conflict with other trains competing for the same resources. Conflict between two trains, train A and train B, will occur if Train B arrives at a timing point before train A has left.

In this model, resolving the conflict involves delaying the arrival and departure time of the train that arrives second (Train B). The alternative would be to resolve the conflict by speeding up the departure of the first train (Train A). However, this would result in trains leaving the station before their scheduled departure time, which would cause disruption and frustration to passengers planning to use that service. The delay added to the second train, to resolve the conflict, is propagated through all of the timing points on the remainder of the train's route. As delaying a train may result in even more conflicts at subsequent timing points, the check for conflict is repeated until all the conflict has been removed from the system.

Table 6.2: Details of delay penalties, in £s, for each train class.

Abbreviation	Train Class	Delay Penalty (£)
XX	Express Passenger	40
XU	Unadvertised Express	40
OO	Ordinary Passenger	20
XZ	Sleeper (Domestic)	20
ZZ	Light Locomotive	20
EE	Empty Coaching Stock	
OTHER	Class not specified in data file	10

The objective is to minimise the delay of all trains within the current evaluation window. The evaluation window determines how far into the future the controller wishes to assess the impact of the platform reallocation decisions. In these experiments it is set to one hour. A train's delay is calculated as in Eq. (6.12) and is the delay at its last timing point within the radius and within the evaluation window, whichever occurs soonest.

$$Z_t = (x_{k,b}^{depart} - d_{k,b}^{initial}) * Penalty \quad k = l_{k,t} \tag{6.12}$$

Where $Penalty$ is the delay penalty, in pounds, for train $x_{k,b}$'s train class. The objective function is to minimise the delay penalty of all station trains within the current evaluation window (Eq. (6.13)). Trains that terminate at Leicester station are not included in the evaluation as they have no ongoing journey.

$$min \sum_{t=1}^{N} Z_t \tag{6.13}$$

The delay penalty for each class of train is shown in Table 6.2. The name for each class code was taken from [125]. The delay penalties reflect the charges levied for train delays on many networks. The high speed trains have the largest penalty in line with [2].

The objective function includes both primary and secondary delay because the resequencing algorithm has the opportunity to reduce dwell time and therefore reduce the primary delay of a train.

The events at each timing point, E_r, includes both events for trains that pass through Leicester station and events for trains that do not pass through the station.

If any non-station trains are delayed due to conflict with station trains at any of the timing points on their route, their delay is also included in the objective function.

6.1.4 Construction of the Leicester Station Model

Much of the current research in rescheduling trains after a delay uses microscopic models which model train movements at the level of detailed infrastructure data, such as block sections, and train dynamics. However, using microscopic models for busy and complex railway networks can result in long computation times for the algorithms used to optimise train schedules after a delay. This makes such models inappropriate for complex and extensive areas of the railway network and makes it difficult to take into account the trains ongoing journey. For this reason this work was based on a macroscopic model created from Network Rail's downloadable file of train schedules [30]. This feed is in JSON format and is an extract of train schedules from Network Rail's ITPS. It is freely available online but requires registration with Network Rail. The use of this data ensures that the model is based on information that is also available to the railway controller making the work suitable for contribution towards a computer-based dispatching system. An advantage of this macroscopic model is that it does not rely on knowing the exact position of each train at every moment in time, via GPS sensors or similar, to work out the impact of the local station decisions on the trains' ongoing journeys. Any new information about the trains' positions could be used to update the train details before running for the next dynamic change period. This would allow solutions in the new change period to be based on the updated positions of the trains.

The data feed contains details about all train schedules over a six-month time period. For each train schedule, it provides an ordered list of the train's route, detailing the arrival and departure times at each timing point on its journey. For the purpose of this work, it is assumed that two adjacent timing points sandwich a block section of track. Only one train can be in a block section at any one time [34, p71]. The assumption made in the creation of this model is that a train is said to depart its current block section once it has departed the timing point at the end of the block section. This is illustrated in Fig. 6.2. The train occupies the track (block) section from the time it enters the timing point at the beginning of the track section (10.00 am), to the time it leaves the timing point at the end of the track section (10.13 am). It therefore occupies the block section from 10.00 am to 10.13 am.

However, this does not allow the length of the train to be taken into account. To prevent conflict and ensure only one train can occupy a track section at a time, the

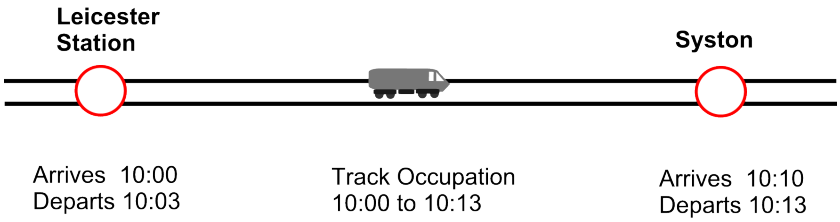


Figure 6.2: Block section occupation

departure time from a block section needs to be the time the end of the train leaves the block section. To calculate this accurately it would be necessary to know the length of each train. This information is not available in the schedule feed therefore an exit margin was added to the departure time of each train from each track section to ensure the end of the train clears the track section. In this work, the exit margin was set to 30 s. This was deemed reasonable because even a very long train of ten carriages and an approximate length of 253 m travelling at only 20 mph (32.1869 kph or 8.94 m/sec) would clear the end of the train in 28.30 s, less than the 30 s exit margin. Eq 6.14 illustrates how the departure time from a track section is calculated.

$$d_{k,b}^{initial} = d_{k,b+1}^{initial} + margin \quad b \in B_r, k \in E_r : k \neq l_{k,b} \tag{6.14}$$

The information needed to model the station's daily operation was extracted from each train's route. The direction the train travels through the station was determined by reference to the timing point the train passes as it leaves the station. Each timing point on a train's route may be used by other trains that may or may not pass through Leicester station. Therefore, the set of arrival and departure events occurring at each timing point on each train's route was also extracted from the data. For the purpose of this investigation schedule data was extracted for Monday 3rd October 2016.

6.1.5 Terminating Trains that become Originating Trains

A discussion with the dispatch team at Birmingham New Street station confirmed that in approximately 99% of cases a train that terminates on a platform will metamorphose into the next service that originates on that platform. An examination of the schedule file confirms this, events that terminate on a platform are usually followed by an originating event on that same platform. The two events have different schedule identifiers but use the same physical train.

Unfortunately, the data feed does not contain exact details of which terminating

events become originating events. Therefore, an assumption was made that if a terminating train is followed by an originating train on the same platform then the events use the same physical train.

It is important to identify metamorphosing events for two reasons:

- If two events use the same physical train it is not physically possible for the events to use different platforms.
- It is not physically possible to insert a different event between two events that use the same physical train.

To deal with terminating events that become originating events the simplest option is to combine them into a single event consisting of the service that terminates at the station and the service it changes into. This prevents the two events from being placed on separate platforms and prevents other trains from being inserted between them. The single event made by combining the originating and terminating events stores the schedule identifiers for both events.

It is of course possible that this way of identifying events that use the same physical trains may miss some metamorphosing events and may incorrectly label other pairs of events as using the same physical train. However, the fact that metamorphosing trains have been identified in the schedule data and that provision has been made for them within the problem formulation shows that the proposed algorithms can handle such events.

6.1.6 Modelling a Train's Ongoing Journey

A train's ongoing journey involves other junctions and stations on its route. When a timing point is another station this is relatively straight forward as the schedule feed gives information about the platform that each train uses at each station. However, the information about the entrance and exit line a train uses when passing through a junction is missing from the schedule feed. To assume that there is only two lines at each junction would give a misleading model and would result in extra conflict in the system. To resolve this, a Track Atlas of Mainland Britain [126] was consulted to identify how many tracks are present at each timing point.

The number of tracks at each junction are recorded in the model and a train that uses a multi-track junction is assigned to one of the lines at that junction. Trains are assigned to lines in the order they arrive at the junction. Conflict is only checked for trains on the same line at a junction. It is recognised that this is a simplistic approach, however, it could easily be refined with more information about the track sections that trains use on the approach and exit to a junction.

6.1.7 Limitations of the Model

Due to lack of accurate data about the movement of trains on the network the model has limitations. For example, assumptions had to be made about which terminating trains became originating trains. In addition due to lack of information about the maximum speed of each train the running time of each train on each track section is the running time extracted from the timetable and is fixed.

These limitations could be resolved by the provision of more descriptive data from Network Rail. However, although the accuracy of the model could be improved, with more information, the model still works as a prototype to investigate train rescheduling after trains are delayed arriving at a station. More data could be used to improve the model but the principle remains the same.

There are a number of aspects of rescheduling that are considered beyond the scope of this current work. The first is that of considering crew duties and rolling stock. Although important, this would introduce additional complexity into the problem, which needs to be tackled as future work. A second limitation is that train connections are not considered in the model. However, with more information they could be integrated into the model to make it a bi-objective problem with the objectives of minimising delay and minimising missed connections. Again this is future work.

6.1.8 Modelling Dynamism

Due to the lack of data concerning real-world delay scenarios and their resolution, delay was introduced to the model by delaying trains at specified time intervals by varying amounts. Dynamism was modelled by adding successive delays at set time intervals. The time intervals between delays represents the frequency of change (f) while the length of delays represents the magnitude of change (m). As trains do not arrive at regular intervals, it is impossible to instigate a delay at an exact time, instead the train nearest to the start of the next change period is the one chosen to be delayed. A delayed train is not only delayed at the station but also at all its scheduled timing points after the station.

The trains delayed for each delay frequency are detailed in tables 6.3, 6.4 and 6.5 respectively.

Table 6.3: Details of the delayed trains for f=10min

Schedule ID	Class	Origin	Destination	Arrival Time	Platform
C65096	HST	TAPTONJ	BEDFDS	7:04	3
C65132	ClassUnknown	BEDFDS	TAPTONJ	7:10	2
C65374	Class150	LOGHBRO	NTNG	7:21	4
C65065	HST	TAPTONJ	BEDFDS	7:33	3
C65031	HST	NTNG	BEDFDS	7:40	3
C60061	ClassUnknown	TAPTONJ	BEDFDS	7:50	4

Table 6.4: Details of the delayed trains for f=20min

Schedule ID	Class	Origin	Destination	Arrival Time	Platform
C65096	HST	TAPTONJ	BEDFDS	7:04	3
C65374	Class150	LOGHBRO	NTNG	7:21	4
C65031	HST	NTNG	BEDFDS	7:40	3

Table 6.5: Details of the delayed trains for f=30min

Schedule ID	Class	Origin	Destination	Arrival Time	Platform
C65096	HST	TAPTONJ	BEDFDS	7:04	3
C65065	HST	TAPTONJ	BEDFDS	7:33	3

6.2 Detection and Resolution of Conflict

The same conflict detection and resolution procedure is used for all algorithms and takes place in two stages, the first stage is to remove all conflict at the station taking into account Network Rail's Timetable Planning Rules [127, page 59] for Leicester station. The second stage resolves the conflict at all of the other timing points on the train's route.

The Timetable Planning Rules for platform reoccupation at Leicester station are as follows:

- For trains travelling in the same direction there must be at least three min between the departure of the first train and the arrival of the second train.
- For trains travelling in opposite directions there must be at least four min between platform reoccupation, unless both trains are HST (High Speed Trains) in which case there must be at least 5 min between trains.

In order to allow the solution provided by the algorithms to conform with current railway practice, the above rules have been implemented when resolving conflict at the station. Before resolving conflict, the directions of both trains are checked along with the train class and the reoccupation margin is determined between two trains using the same platform. The reoccupation margin is enforced between two trains by adjusting the arrival time of the second train.

A set of feasible route timings is conflict free if for every pair of trains that requires the same block section, the departure time of the train that arrives first (x_{k-1}^{depart}) is less than the arrival time of the train that arrives second (x_k^{arrive}). That is the first train to use the block section departs before the next train arrives to use it.

Conflict is detected and resolved for one train at a time according to the departure order dictated by the train schedule (for the reallocation sub-problem) or the solution order (for the resequencing sub-problem). If a train has a delay, then the reallocation problem takes into account the new departure order of the train.

The first step in the conflict detection is to identify trains that arrive or depart within the time window of the train currently under investigation. Fig. 6.3 illustrates this. The left side of the box surrounding the train indicates the train's arrival, the right side of the box shows the train's departure. The train in red is the train currently being assessed for conflict. It arrives at the timing point at 9.30 and departs at 9.50. Any trains that depart before the red train arrives or arrive after the red train departs, the grey trains in this figure, will not come into conflict with

the red train and therefore are excluded from the conflict detection in order to reduce computation time.

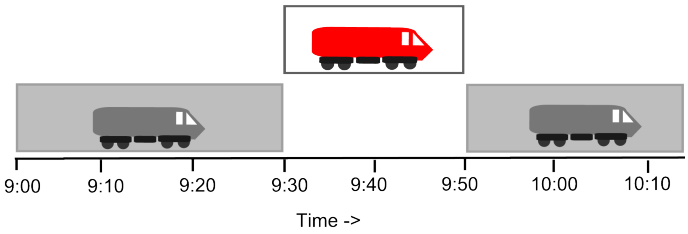


Figure 6.3: Grey trains outside the time window of the red train

Fig. 6.4 shows a conflict between two trains. In this case train $T2$ has not left the track section before train $T1$ arrives. $T2$ departs at 9.30 am while $T1$ arrives at 9.18 am. The overlap between them is shown as a hatched box and can be determined by subtracting $T1$'s arrival time from $T2$'s departure time. In this case it is 12 minutes.

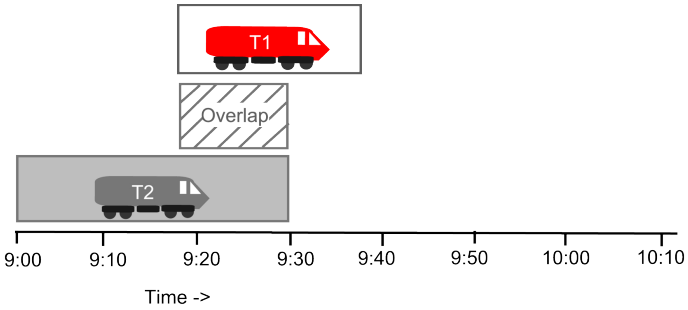


Figure 6.4: Conflict caused by $T1$ arriving before $T2$ has left

Fig. 6.5 shows the new track occupation time for $T1$. It has been delayed so that it now arrives after $T2$ has departed.

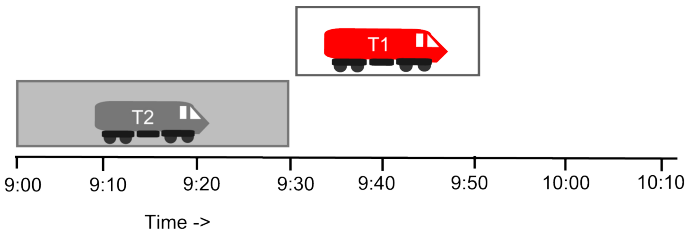


Figure 6.5: Resolution of conflict between $T1$ and $T2$ by delaying $T1$

Fig. 6.6 shows the situation where $T2$ arrives before $T1$ has left. $T2$ arrives at 9.11 am while $T1$ does not leave until 9.20am. In this case $T2$ is delayed to remove the overlap between the two trains so that $T2$ now arrives after $T1$ has left (see Fig. 6.7).

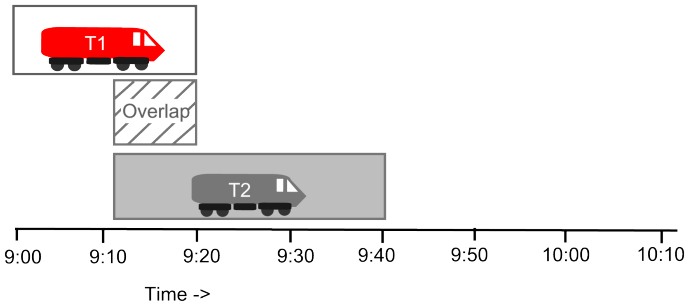


Figure 6.6: Conflict caused by $T2$ arriving before $T1$ has left

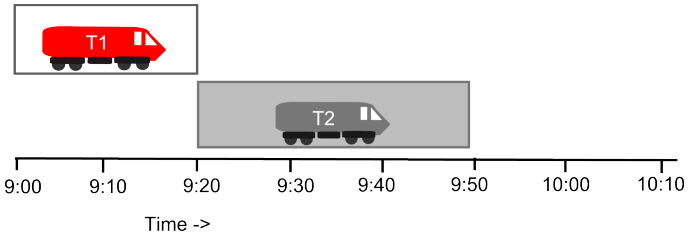


Figure 6.7: Resolution of conflict between $T1$ and $T2$ by delaying $T2$

Once conflict has been detected it is resolved by delaying the second train to arrive at the track section. A margin of 30 s is inserted between the departure of the first train and the arrival of the second train as detailed in Section 6.1.4.

The reason that conflict is resolved by delaying the second train to arrive is because this is the solution that will result in the least disruption. Making the first train depart earlier to remove the conflict would mean that trains may depart earlier from the station than their scheduled departure time. This would cause problems for train crew and passengers. In addition delaying the arrival of the first train instead of the second train would mean that the first train's arrival would have to be moved back before the arrival of the second train. If there are no suitable passing places, the first train would have to leapfrog over the second train to be able to arrive after it. This would be infeasible.

The delay added to avoid conflict is propagated through all of the timing points on the remainder of the trains journey. Conflict detection and resolutions takes place in a loop as delaying one train may cause knock on delays to other trains. The delay resolution is repeated until all the delay has been removed from all of the timing points on all of the routes for all of the trains in the current time windows.

A check for conflict is made at each timing point on a train's route. The check is made with all other trains at that timing point in the same time period. The conflict check is made in the order that the timing points appear in the train's route. Any delay added to a train to avoid conflict with another train is only propagated

Algorithm 3 Conflict Detection and Resolution

```
1: Input T1                                          ▷ Train under investigation
2: Input T2                                          ▷ A potentially conflicting train
3: Input E^TP                                        ▷ Set of events at one timing point
4: Input n^e                                         ▷ Number of events at one timing point
5: Input n^tp                                        ▷ Number of timing points on a trains route
6: Input flag=true                                   ▷ Set to true if confict detected
7: Input overlap  ▷ Difference between the departure of one train and the arrival
   of the next train
8: Input margin  ▷ Margin added to the overlap to ensure the train has cleared
   the track section
9: while flag is true do
10:     flag = false
11:     for (i=0 to i=n^tp-1) do                     ▷ for each timing point
12:         for (j=0 to j=n^e-1) do                  ▷ for each event at the timing point
13:             T2 = E_j^TP
14:             if T2 is within T1's time window then
15:                 if T1 arrives before T2 has left then
16:                     overlap = T2 departure - T1 arrival +margin
17:                     Delay T1
18:                     Propagate Delay along remainder of T1's route
19:                     flag = true;
20:                 end if
21:                 if T1 arrives before T2 has left then
22:                     overlap = T1 departure - T2 arrival +margin
23:                     Delay T2
24:                     Propagate Delay along remainder of T2's route
25:                     Mark T2 as altered
26:                     flag = true;
27:                 end if
28:             end if
29:         end for
30:     end for
31: end while
```

forwards along the train's route, never backwards. The assumption is made that delaying a train at a timing point means that the train arrives later at the timing point, this is achieved by the train slowing down and increasing the time taken to traverse the section of track before the timing point in question.

To reduce computation time the check for conflict is only made if the train has been changed in any way by the rescheduling process, for example if it has had primary or secondary delay added or its platform has been changed.

The implementation of the conflict detection and resolution procedure is detailed in Algorithm 3.

6.2.1 The Reallocation and Rescheduling Sub-Problems

As previously mentioned the problem of rescheduling trains after a delay at a station can be divided into the two sub-problems of reallocation and resequencing. Reallocation assigns a suitable platform slot for the train while resequencing determines the order the train leaves the station. Breaking the rescheduling problem into two sub-problems is a common approach in railway research, for example, Corman *et al.* [54] rerouted trains with a TS algorithm and then rescheduled them, while Samà *et al.* [23] used MMAS to find an effective set of train routes which were then used as input to a MILP rescheduling module.

In this work, the rescheduling of trains at stations is broken down into two stages for two reasons. The first reason is that the format of the solution for each part of the problem is different. In the case of reallocation, the solution consists of a train and a platform, while in the resequencing problem the solution consists of a list of trains in the order they are to leave the station. In fact the reallocation part of the problem can be thought of as a job shop scheduling problem (JSP) while the resequencing part can be thought of as a travelling salesman problem (TSP).

Secondly, attempting to use one ant colony to perform both halves of the task would create an extremely large directed edge graph as an ant would have to assign each train to a platform and then to a departure order. Each train can have a choice of four platforms and four departure orders making a total of 16 choices for each train. With approximately 30 trains in the each dynamic change this would make a graph that includes 480 nodes. In contrast, using two colonies results in two graphs containing 120 nodes (30 trains x 4 platforms) for the first colony and 30 nodes (one for each train) for the second colony. The larger the graph, the larger the search space and the more ants may be needed to explore it. As the computation time increases with the number of solution evaluations then this will result in a corresponding increase in the execution time.

Finally combining the colonies makes it very difficult to decide how to assign a departure order to a train. This is a consequence of the fact that they are very different categories of problems. If a platform is assigned to a train and then a departure order, the departure order will depend on the other trains in front of that train on the platform, which depends on the platforms allocated to the trains by the ants. This will make the pheromone information for the colony very complicated and difficult to interpret as it will depend on the platforms previously chosen by the ants.

When evaluating the solutions produced for each sub-problem, the format of the sub-problem has to be taken into account. This means that the evaluation of a

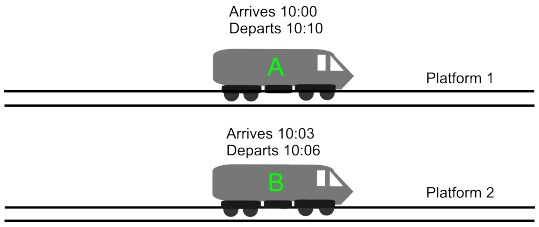


Figure 6.8: Trains A and B waiting on a platform before a delayed train is reallocated to a platform.

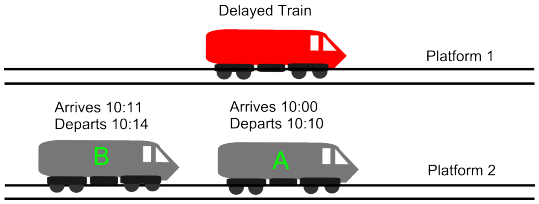


Figure 6.9: The situation after a delayed train has caused trains to be reallocated to new platforms.

reallocation solution is different to the evaluation of a resequencing solution. The specific method of evaluation for each sub-problem is described in the following sections.

6.2.2 Evaluation of a Reallocation Solution

To evaluate a reallocation solution each train is assigned to its allocated platform and the platform displacement for that solution is calculated. This change is made in a copy of the events at the station to ensure that each ant works on data unchanged by a previous ant. Any train that has been assigned a new platform is marked as altered to ensure it will be checked for conflict with all other trains on that platform.

The events at the station are sorted in ascending departure order. In this case the departure order is the original departure order of the trains taken from the original train schedule. The conflict for all trains is resolved one at a time in the order that the trains depart from the station as described in Section 6.2.

However, even though there is a separate sub-problem that changes the order the trains leave the station, the reallocation problem itself can indirectly affect the departure order. Fig. 6.8 shows two trains at the station, train A on platform 1 and train B on platform 2. Although train A arrives before train B its departure time is later, therefore train B will depart before train A. Fig. 6.9 shows what could happen if a delayed train is assigned to platform 1. Train A is displaced to platform 2. It arrives before train B so it is placed before train B on the platform. To allow this to happen the arrival of train B has to be delayed to ensure it is after train A departs.

This also delays train B's departure which means that train B now leaves after train A. In this case the reallocation has indirectly changed the order that train A and train B leave the station.

6.2.3 Evaluation of the Resequencing Solution

When evaluating a resequencing solution, a check has to be made to ensure that the track ahead is clear of any other trains before allowing a train to leave the station. To achieve this the last departure time from the relevant direction is recorded. This departure time is the time the train clears the timing point after the station thus ensuring that the path is clear for the next train.

The departure time for the next train scheduled to leave the station ($x_{k,b}^{depart}$) is compared with the last departure from the station in the same direction ($x_{k-1,b}^{depart}$). If $x_{k,b}^{depart}$ is less than $x_{k-1,b}^{depart}$, $x_{k,b}^{depart}$ cannot be allowed to leave yet as it will conflict with the train that left before it. For example, if the previous train clears the next timing point after the station at 10 am but the current train would reach the timing point at 9.30am then the current train cannot be allowed to leave yet and must be held in the station by increasing its dwell time. Increasing the dwell delays the train's exit from the station and the delay is propagated through all of the timing points on the remainder of the train's route.

However, if $x_{k,b}^{depart}$ is later than $x_{k-1,b}^{depart}$. For example, the last departure reaches the next timing point at 9.30am and the current train would leave at 10am then there is the opportunity to allow the current train to leave earlier by reducing the dwell time. However, the dwell time cannot be made less than the minimum dwell time for that category train, as detailed in the rules of the plan [127, page 58] and the train cannot be allowed to leave earlier than its original departure time [54]. Algorithm 4 details the steps taken to evaluate a resequencing solution.

If a train has zero dwell it means it will pass through the station without stopping. In this model, such a train is allowed to be held at the station if that improves the overall minimisation delay in the system. This allows for overtaking of such trains at the station. This option could of course be changed if discussion with the railway dispatcher revealed that that is not their policy and that trains that pass straight through must not be stopped.

6.2.4 After a Change

One of the aims of this work is to model a real-world dynamic railway rescheduling problem, therefore, it is necessary to consider how it would work in a real-life

Algorithm 4 Evaluating a Resequencing Solution

1: **Input** S ▷ A resequencing solution
2: **Input** n ▷ Number of trains in S
3: **Input** $prevDeparture$ ▷ The previous departure from the station in this direction
4: **Input** $originalDeparture$ ▷ Original departure time
5: **Input** $minimumDwell$ ▷ Minimum dwell time
6: **Input** $newDeparture$ ▷ Updated departure time
7: **Input** $newDwell$ ▷ Updated dwell time
8: **for** `(i=0 to i=`$n-1$`)` **do** ▷ for each train in the tour
9: **if** $x_{k,b}^{depart} \leq prevDeparture$ **then**
10: Hold $x_{k,b}$ in the station by increasing dwell
11: Propagate additional delay along remainder of $x_{k,b}$'s route
12: **else if** $x_{k,b}^{depart} > prevDeparture$ **then**
13: **if** $newDwell \geq minimumDwell$ and $newDeparture \geq originalDeparture$ **then**
14: Allow $x_{k,b}$ to depart earlier by reducing dwell
15: Propagate delay reduction along remainder of $x_{k,b}$'s route
16: **end if**
17: **end if**
18: $prevDeparture = x_{k,b}^{depart}$
19: **end for**

situation. The assumption is made that once an optimal solution to the problem is found it is implemented by the traffic controller. The solution runs until the next change. When the next change occurs, the state of the station at that point in time is used to identify trains that have passed through the station and are no longer of relevance to the problem and new trains that are now within the problem boundary. This means that the longer the interval between changes the more trains will have passed through the station and will no longer be of relevance to the algorithms.

When a change occurs the start (sp_c) and end times (ep_c) of the problem are recalculated as in Eqs. (6.7) and (6.11). The best-so-far solution for the previous change period ($c-1$) is implemented and run to update the arrival times, departure times and platforms of all trains in the previous and the current time period. This essentially creates a snapshot of the problem at the point of change. The updated arrival and departure times are used to establish if the trains fall within the updated problem window for change c. Any trains that arrive within the problem window are added to the problem, any that arrive before the start of the problem window are removed from the problem and can no longer be altered at the station. Trains that have been removed from the problem are not passed to the algorithm, but any new trains that fall into the time window for current change period are passed to

the algorithm to be included in the new problem.

6.3 ACO for Dynamic Optimization Problems (DOPs)

In this work, the performance of three different ACO algorithms are compared. The three algorithms investigated are P-ACO developed by Guntsch and Middendorf [36]; MMAS developed by Stützle and Hoos [37]; and ACS developed by Dorigo and Gambardella [39, 40]. Details of these algorithms can be found in Section 2.2.2, Section 2.2.3 and Section 2.2.4 respectively.

These three versions of the ACO algorithm were chosen as they are ones commonly observed in current ACO research papers. This suggests that they have consistently good, research-worthy, performance. In fact, MMAS is considered a 'state-of-the-art' ACO algorithm [128]. P-ACO is seen less frequently, however, it was included because its inbuilt memory makes it very suitable for dynamic optimisation problems as information from before the change is automatically carried over to after the change. In addition, P-ACO has previously been applied to the dynamic TSP [129] and a dynamic junction rescheduling problem (see Chapter 4) with good results.

A final reason for the comparison of these three algorithms is that the mode of operation for each is very different. P-ACO holds a memory of previous solutions that are used to update the pheromone values. MMAS starts with initial high pheromone values and then reduces them over time, proportional to the performance of the best ant. ACS updates the pheromones with both a local and a global update. Applying algorithms with different modes of operation may make it more likely to find an algorithm that works well on this specific problem.

6.3.1 Related Work Using ACO for Dynamic Rescheduling

As previously mentioned in Section 6.2.1, the reallocation sub-problem, of the DSRP, can be considered to be a dynamic JSP, while the resequencing sub-problem can be considered to be a dynamic TSP similar to the DRJRP in Chapter 4. Related work concerning the application of ACO to the dynamic TSP has already been described in Section 3.6. In this section, related literature concerning the application of ACO to dynamic JSP problems is considered.

For a JSP, the aim is to assign jobs to machines at particular time slots. In the reallocation sub-problem, the jobs correspond to trains and the machines correspond

to station platforms. Previous work in applying ACO to dynamic JSPs has shown promising results. Both Xiang and Lee [130] and Renna [131] combined ant colony intelligence with a multi-agent system (MAS) to solve a dynamic JSP. Xiang and Lee found that the MAS with ant colony intelligence outperformed a MAS with a first in first out (FIFO) dispatching rule, while Renna found that the ant intelligence approach gave a solution that was comparable with a coordination approach when the dynamic changes were of low or medium frequencies. Zhou *et al.* [132] applied ACO to a dynamic JSP and found that it outperformed a heuristic based on the shortest processing time (SPT) while Lu and Romanowski [133] found that their version of ant colony system (ACS) outperformed well-known dispatching rules such as FIFO and SPT. This research suggests that ACO may show good performance when applied to the dynamic reallocation sub-problem of the DSRP.

6.3.2 The Reallocation and Resequencing Colonies

One of the goals of this work is to investigate the effectiveness of combining two different ant colonies to carry out the reallocation and resequencing tasks. One colony (COLONY-R) reallocates trains to platforms while a second colony (COLONY-S) resequences trains to determine the order that the trains should leave the station.

In the following sections the colonies used for reallocation and resequencing are described in detail. Each colony uses the same detection and resolution of conflict for the trains at the timing point described in Section 6.2. The difference between them is the format of the tour provided by the ant and the way the tour is interpreted. The interpretation of the tour for each ant colony is described below.

COLONY-R

COLONY-R determines the platforms to be used by each train in the problem. The solution made by an ant from COLONY-R consists of a list of trains with an associated platform. The ants can potentially assign a new platform to every train in the problem.

To apply ACO to an optimization problem, it has to first be decomposed into a fully connected weighted graph $G = (V, E)$, where V is a set of vertices or nodes and E is a set of edges or connections between the nodes. In this problem, each node represents a train on a platform. For example, node 1 represents train A on platform 1, node 2 represent train A on platform 2, etc (see Fig. 6.10). Ants move from node to node recording the nodes visited. Once they have selected a platform for one train they move on to the next train. At the end of their tour, they have

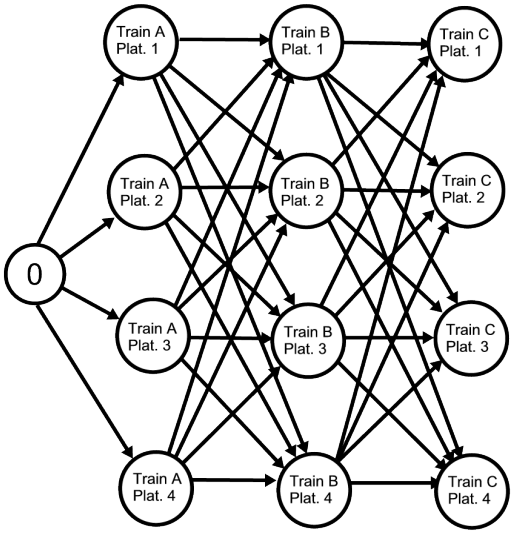


Figure 6.10: The directed edge graph the ants use to construct their tour. Each circle represents a node.

constructed a solution that consists of a list of trains and the platforms they have been allocated to. At the end of the iteration, the best ant deposits pheromones on the edges of the graph, reinforcing its choices for the next iteration of ants. Ants are presented with trains in the order they arrive at the station. This is to prevent them from creating an infeasible solution in which a train is placed on a platform in front of a train that arrives before it.

After a delay some trains will have been removed from the problem as they will have arrived at the station and it will be too late to change their platform. These trains are no longer of interest to the ants. Therefore, they are removed from the directed edge graph that the ants move around to construct their tour, and are also removed from the pheromone matrix. However, new trains will be approaching the station and will be included in the new problem passed to the ants. These new trains are added both to the directed edge graph and to the pheromone matrix.

The outcome of this algorithm is a list of trains and their associated platforms. The list is passed to the reallocation evaluation function described in Section 6.2.2 that decodes the solution and resolves any conflict at the station, and on the trains' ongoing journeys, that results from the reallocation decisions.

COLONY-S

COLONY-S determines the order the trains leave the station. A solution made by an ant from COLONY-S consists of a list of trains in the order they are to depart the

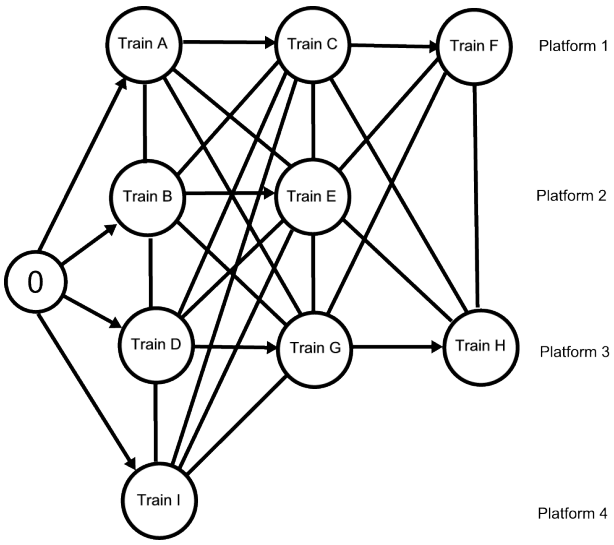


Figure 6.11: The directed edge graph the ants in COLONY-S use to construct their tour. Each circle represents a train. The horizontal lines represent platforms.

station. As there are four platforms that feed into one exit line there is a potential for up to four trains to be waiting to leave the platform on the same exit line. This algorithm decides which of those trains should depart next. This means that a train that has a higher delay penalty has the potential to be allowed to leave first which may reduce its ongoing delay penalty.

In this problem the directed edge graph has a different format to COLONY-R and its design ensures that the ants cannot make infeasible solutions where a train is assigned to leave the station before a train that is before it on the same platform.

Fig. 6.11 shows the directed edge graph for the resequencing sub-problem. In this case each node represents a train. Each horizontal row in the graph represents a different platform. To prevent infeasible solutions the trains are placed on the platforms in the order they arrive at the station. An ant moves along the graph from left to right selecting the next train to leave the station. The arrows between trains on the platform indicate that the ant can only move one way and therefore cannot select train C before train A, for example. Node 0 represents the start node. At the beginning of each iteration, all ants are placed on this node. After a change, the graph may shrink or grow depending on the number of trains added and the number of trains removed from the problem because they have passed through the station and are no longer relevant.

In this sub-problem, there is no computationally efficient and effective problem-specific heuristic available. Therefore, the ants rely purely on the pheromone values for guidance.

The outcome of this algorithm is a list of trains in the order they should depart

Algorithm 5 Multi-colony algorithm MC1

```
1: Input BS-R                                          ▷ Best solution Colony-R
2: Input BS-S                                          ▷ Best Solution Colony-S
3: Input BS                                  ▷ Best Solution over both colonies
4: Input trains                              ▷ Trains in current time window
5: Input c                                             ▷ The number of changes
6: Initialise Colony-R
7: for (i=0 to i=c − 1) do                                    ▷ for each change
8:     while (termination condition not satisfied) do
9:         run Colony-R
10:    end while
11:    BS-R = ColonyR.bestSolution          ▷ Store best solution for Colony-R
12:    Update arrival order of trains based on BS-R
13:    Initialise Colony-S
14:    while (termination condition not satisfied) do
15:        run Colony-S
16:    end while
17:    BS-S = ColonyS.bestSolution          ▷ Store best solution for Colony-S
18:    if BS-S.fitness < BS-R.fitness then
19:        BS = BS-S
20:    else if BS-R.fitness ≤ BS-S.fitness then
21:        BS = BS-R
22:    end if
23:    if change occurs then
24:        Reinitialise Colony-R using BS for snapshot
25:    end if
26: end for
```

from the station. The list is passed to the evaluation function described in Section 6.2.3 that determines the times the trains leave the station to ensure there is no conflict between any of the trains when they depart.

6.4 Framework for Combining the Two Colonies

One of the aims of this work is to investigate the effectiveness of combining two colonies of ants to perform both reallocation and resequencing at the station. One of the issues that arise is whether to carry out reallocation followed by resequencing or resequencing followed by reallocation. Most of the existing research in railway rescheduling, for example [54, 10], carries out route allocation followed by rescheduling, which suggest that this may be the best option. However, to discover which method performs best, for this problem, two multi-colony algorithms have been developed. The first (MC1) carries out reallocation followed by resequencing, the

Algorithm 6 Multi-colony algorithm MC2

```
1: Input BS-R                                          ▷ Best solution Colony-R
2: Input BS-S                                          ▷ Best Solution Colony-S
3: Input BS                                  ▷ Best Solution over both colonies
4: Input trains                              ▷ Trains in current time window
5: Input c                                             ▷ The number of changes
6: Initialise Colony-S
7: for (i=0 to i=c − 1) do                                      ▷ for each change
8:     while (termination condition not satisfied) do
9:         run Colony-S
10:    end while
11:    BS-S = ColonyS.bestSolution          ▷ Store best solution for Colony-S
12:    Update departure order of trains based on BS-S
13:    Initialise Colony-R
14:    while (termination condition not satisfied) do
15:        run Colony-R
16:    end while
17:    BS-R = ColonyR.bestSolution          ▷ Store best solution for Colony-R
18:    if BS-S.fitness < BS-R.fitness then
19:        BS = BS-S
20:    else if BS-R.fitness ≤ BS-S.fitness then
21:        BS = BS-R
22:    end if
23:    if change occurs then
24:        Reinitialise Colony-S using BS for snapshot
25:    end if
26: end for
```

second (MC2) carries out resequencing followed by reallocation. Algorithm 5 describes MC1, while Algorithm 6 describes MC2.

In both of these multi-colony solutions there is only one run of the first colony followed by a single run of the second colony. It is possible that repeated iterations of the two colonies would give improved results, however, it would also result in an extremely long execution times which would not be feasible in a real-world rescheduling situation. Dollevoet *et al.* [66] found that an iterative approach that involved running two algorithms one after each other until a good solution was found took too long to be considered suitable to solve a real-time rescheduling problem.

6.5 Reducing Unnecessary Platform Reallocation

Initial experimentation revealed that, as ACO has no inbuilt intelligence to persuade it against unnecessary reallocation of trains to new platforms, the algorithm would

often change a train's platform if the change had no impact on the fitness of the solution. In the real world, this would result in disgruntled passengers and unhappy railway employees. To solve this problem, a platform displacement heuristic (η_{ij} in Eq. (2.1)) was introduced to guide the ants in making intelligent platform reallocation choices. The heuristic takes into account the fact that Leicester station has two sets of two platforms separated by stairs. Platforms 1 and 2 are conjoined as are platforms 3 and 4. Moving between either of these is a simple case of crossing from one side of the platform to the other. However, moving from platform 1 or 2 to platform 3 or 4 involves negotiating a set of stairs. The heuristic ensures that the desirability of moving from stair-linked platforms is much lower than that of moving from conjoined platforms and that the desirability of both is much lower than leaving the train on its original platform. The heuristic is given by $1/PD$ where PD is a representation of the physical distance between the current node's platform and the decision node's platform. PD is 1 if the platforms are the same, 2 if they are conjoined, or 4 if they are separated by a set of stairs.

To reinforce the heuristic, a novel method was created which decides whether to replace the best-so-far ant (ant^{bs}) with the best iteration ant (ant^{bi}), after each iteration. This method takes into account both the solution's objective value and the amount of platform displacement. Experiments detailed in Appendix A show that this approach works well to reduce unnecessary platform reallocation therefore this novel best-so-far ant replacement schemes was used for the all the algorithms applied to the reallocation sub-problem.

6.6 Comparison Algorithms

Originally a heuristic based on finding the first free platform as close as possible to the delayed train's original platform was considered for comparison. Discussions with a Network Rail Station Master established that this technique is often used to reallocate delayed trains to platforms as it minimises passenger and crew disruption. However, it was found that often it was not possible to find a suitable gap for a delayed train, especially when trying to obey the timetable planning rules detailed in Section 6.2. Appendix B shows the algorithmic details of this heuristic and gives the results of running it with different frequencies and magnitudes of delay. As can be seen, in many cases no suitable free platform could be found.

Therefore, the performance of the ACO algorithms was compared with two algorithms based on local search, the TS and the VNS. Both these algorithms have previously been applied to the railway rescheduling problems. Corman *et al.* [54]

applied a TS while Samà *et al.* [10] applied a VNS. To make the comparison fair, in both TS and VNS the same novel best-so-far ant selection method detailed in Section 6.5 is used when selecting the solution to replace the incumbent. Both of these algorithms require the use of a local search technique, which is described in Section 6.6.1 for the reallocation sub-problem and Section 6.6.2 for the resequencing sub-problem.

6.6.1 Local Search for the Reallocation Problem

In the reallocation problem the local search solutions are created by randomly changing the platform for each train in the set of trains passed to the algorithm. The trains are taken in turn, so that first train 1 is assigned a new platform to make local search solution one, then train 2 is assigned a new platform to make local search solution two. This is repeated for the number of trains in the current problem (N) giving a total of N local search solutions.

6.6.2 Local Search for the Resequencing Problem

The issue with making local search solutions for the resequencing problem is that it is important to create a set of feasible local search solutions that ensure a train is not scheduled to leave the station before the train before it on the same platform. For this reason the solutions were created using the path-preserving local search described in Section 4.2.2. The technique runs a forward pass through the list of trains and then a backward pass to obtain a reshuffled train order that is guaranteed to be feasible.

6.6.3 Tabu Search (TS)

The procedure for the TS is shown in algorithm 7. In this case the incumbent, or best-so-far, solution is compared with the best solution found in the neighbourhood of the incumbent solution. If the neighbourhood solution is an improvement it replaces the current incumbent. Solutions that have been previously considered by the algorithm are placed in a tabu list so that they are not used again.

Identifying if a solution exists in the tabu list can be computationally expensive. To reduce computation time the solutions are first matched with the tabu list solutions on fitness. If there are no solutions in the tabu list that have the same fitness as the comparison solution then the comparison solution is not present in the tabu list. However, if there are solutions that match on fitness they are investigated further. In this case the components of each solution (each train and platform)

Algorithm 7 Tabu Search

```
1: Input neighbourhood                          ▷ local neighbourhood search solutions
2: Input n                                      ▷ number of solutions in neighbourhood
3: Input tabuList                               ▷ list of already visited solutions
4: Output incumbent                             ▷ the current best Solution
5: incumbent ← selectRandomSolution(neighbourhood)
6: while (termination condition not satisfied) do
7:     Input bestCandidate
8:     for (i=0 to i=n-1) do
9:         if (neighbourhood[i].fitness)<incumbent.fitness)and(neighbourhood[i]
   not in tabuList) then
10:            bestCandidate ← neighbourhood[i]
11:        end if
12:    end for
13:    if (bestCandidate.fitness)<(incumbent.fitness) then
14:        incumbent ← bestCandidate
15:    end if
16:    tabuList.push(bestCandidate)
17:    if (tabuList.size > maxTabuSize) then tabuList.removeFirst()
18:    end if
19: end while
```

are compared to the solutions in the tabu list. As soon as a component is found that does not match the comparison solution it can be assumed to be different to that solution in the tabu list. However, if every component of the tabu list and the comparison solution is the same, the comparison solution is present in the tabu list and is discarded.

After a change neighbourhood local search solutions are recreated using the trains in the current time window and the tabu list is cleared ready for the new problem.

6.6.4 Variable Neighbourhood Search (VNS)

VNS was introduced by Mladenović and Hansen [134]. The algorithm uses the concept of changing the neighbourhood during the search to avoid becoming trapped in a local optimum. It explores each neighbourhood in turn, if it does not find a solution in the current neighbourhood that improves the current incumbent solution it moves onto the next neighbourhood. The neighbourhoods are designed to be increasingly distant from the current incumbent solution.

The procedure for VNS is shown in algorithm 8. To begin, the neighbourhoods are created and a solution is chosen randomly from the first neighbourhood to become the current incumbent solution. The algorithm then cycles through each of

Algorithm 8 VNS

```
1: Input neighbourhoods                          ▷ the local search neighbourhoods
2: Input k                                       ▷ number of neighbourhoods
3: Output incumbent                              ▷ the current best Solution
4: incumbent = selectRandomSolution(neighbourhoods[0])
5: while (termination condition not satisfied) do
6:     for (i=0 to i=k-1) do
7:         randomSolution←selectRandomSolution(neighbourhoods[k])
8:         candidate ← localSearch(randomSolution)
9:         if (candidate.fitness)<incumbent.fitness) then
10:            incumbent←candidate
11:            i ← k;                                ▷ exit the loop
12:        end if
13:    end for
14: end while
```

the neighbourhoods in turn picking a random solution, performing a local search and if the best solution from the local search is better than the current incumbent solution, it replaces the incumbent with this solution. If the incumbent is replaced with the best solution then the algorithm does not visit any of the other neighbourhoods. However, if a better solution is not found in the first neighbourhood, the algorithm moves on to the next neighbourhood. If a better solution is still not found, the algorithm continues visiting increasingly distant neighbourhoods until a new solution is found or there are no more neighbourhoods.

The local search for the reallocation problem is the same as that described in section 6.6.1, while the local search for the resequencing problem is that described in section 6.6.2.

The difference between VNS and TS is that TS uses only one neighbourhood structure. Using VNS raises the question of how the neighbourhoods should be created. The following sections describe the neighbourhood creation process for the reallocation and resequencing sub-problems.

Neighbourhood for the Reallocation Problem

In the reallocation problem the distinction between the neighbourhoods is based on how many trains in the current solution are assigned new platforms. If only one train is assigned a new platform, then this solution will be placed in neighbourhood one. If two trains are assigned a new platform the solution will be placed in neighbourhood two, and so on. The more trains that are assigned new platforms, the further away the new solution is from the original solution and the further away the allocated neighbourhood.

Neighbourhood for the Resequencing Problem

The neighbourhoods for the resequencing problem are created using the path-preserving local search described in Section 4.2.2. In this case which neighbourhood the train is placed in depends on the number of trains that are swapped with other trains. If only one train is swapped with other trains, then this solution is placed in neighbourhood one. If two trains are swapped with other trains, then this train is placed in neighbourhood two, and so on.

It is possible that there may be no feasible solutions in one or more neighbourhoods. If this is the case, then that neighbourhood is skipped and the algorithm moves onto the next neighbourhood.

The neighbourhoods are based on the current incumbent [134]. Therefore, at the end of an iteration the neighbourhoods are recreated based on the incumbent solution. This occurs for the reallocation sub-problem whether or not the incumbent has been replaced because randomly generating new platforms for set numbers of trains may result in a different neighbourhood in the next iteration. However, if the incumbent has not been replaced for the resequencing sub-problem, the neighbourhood will not be remade as there would be no change in the local solutions within the neighbourhoods produced from the unchanged incumbent.

After a dynamic change the neighbourhoods are recreated using the trains in the current time window.

6.7 Experimental Study

Experiments were carried out in two stages. In the first stage, the individual sub-problems are investigated in order to see how well the algorithms performed on each sub-problem. In the second stage the performance of the multi-colony approach is evaluated.

The parameters for each algorithm were pre-determined by experimentation and are detailed in table 6.6.

In the version of MMAS used in the DSRP, the ant used to update the pheromones was chosen in a ratio of 2:1 of the best-so-far ant to the best iteration ant. For two iterations, the best-so-far ant was chosen followed by one iteration where the best-iteration ant is chosen. This was found to give the best performance in preliminary investigations.

In all cases the algorithm was terminated when there had been no change in the best-so-far solution for 20 iterations or when 500 iterations had been performed. 30 runs were executed for each dynamic scenario.

Table 6.6: Parameter Settings for the DSRP

Algorithm	Num. Ants	α	β	q0	Algorithm Specific Parameters
P-ACO	50	1.0	1.0	0.9	Memory size=6
MMAS	50	1.0	1.0	1.0	a=10, $\rho = 0.5$, Reinitialisation Interval=20
ACS	50	1.0	1.0	0.9	Global update: $\rho = 0.1$, Local update: $\xi = 0.1$
TS					Maximum size of tabu list = 20
VNS					Maximum number of neighbourhoods=5

There are two interesting aspects of train delays from the viewpoint of the railway controller. The first is that of how much notification the controller has about the delay (the notification period), the second is that of how far into the future the controller wishes to look when considering where to place a delayed train (the planning horizon). Experiments were carried out in previous work [123] to investigate the effect the notification and planning periods have on the performance of the algorithm. These are reproduced in Appendix C.

It was found that a planning horizon of 60 min gave a slightly better performance than a short planning horizon of 10 min. This is to be expected as the longer the planning horizon the more trains can be included in the problem given to the ants and the more options they have to rearrange those trains on the platforms to reduce the objective value. In contrast, the notification interval appears to have very little influence on the performance of the algorithm which suggests that rearranging trains that arrive before the delayed train has very little influence on the outcome of the algorithm and that it is trains that arrive later than the delayed train that are the important ones in terms of finding a good problem solution.

In all the following experiments, a planning horizon of 60 mins was used to give the best possible outcome. This is in line with Törnquist [59] who found a planning horizon of 60 minutes was sufficient to give good long-term results. As already mentioned the notification period appears to have very little influence on this problem and so a value of 30 min was chosen.

In the following experiments the performance of each of the ACO algorithms is compared with the performance of TS, VNS and with running without any reallocation or resequencing (NO-ALG). The first set of experiments considers the performance of the algorithms on each of the sub-problems. The second considers the results of combining two colonies of ants that work together to solve the DSRP.

6.7.1 Experimental Results for each of the Sub-problems

Nine different dynamic environments were investigated involving all permutations of 3 different magnitudes of change (10 mins, 20 mins, 30 mins) and 3 different change frequencies (10 mins, 20 mins, 30 mins). All delays occurred over a period of one hour and started at 7am.

The next section considers the results in terms of average total delay penalty, while the next-but-one section considers the results in terms of average platform displacement.

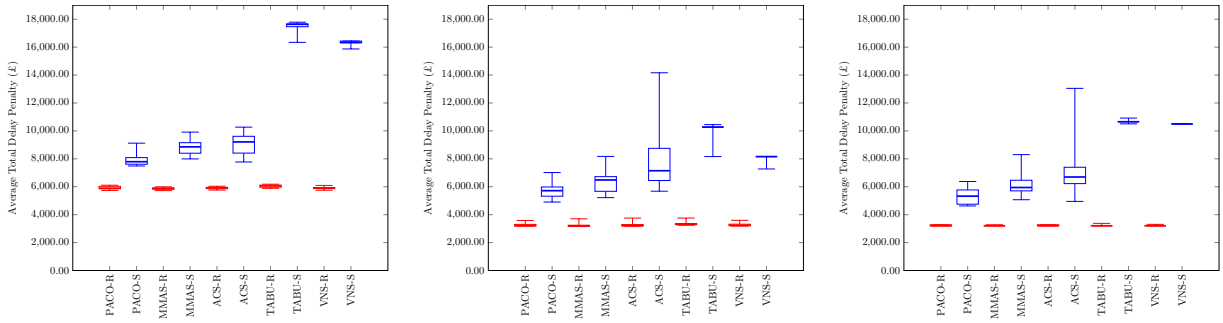
Average Total Delay Penalty

Fig. 6.12 shows box plots for the comparison of total delay penalty averaged over all changes for each algorithm. The algorithm names are listed along the bottom of each boxplot. The box plots in red show the results using reallocation only, in this case each algorithm name is suffixed with a R. The box plots in blue show the results using resequencing only, in this case each algorithm name is suffixed with a S. PACO refers to population based ACO; MMAS to the Max-Min Ant System algorithm; ACS to Ant Colony System; TABU to tabu search; and VNS to variable neighbourhood search.

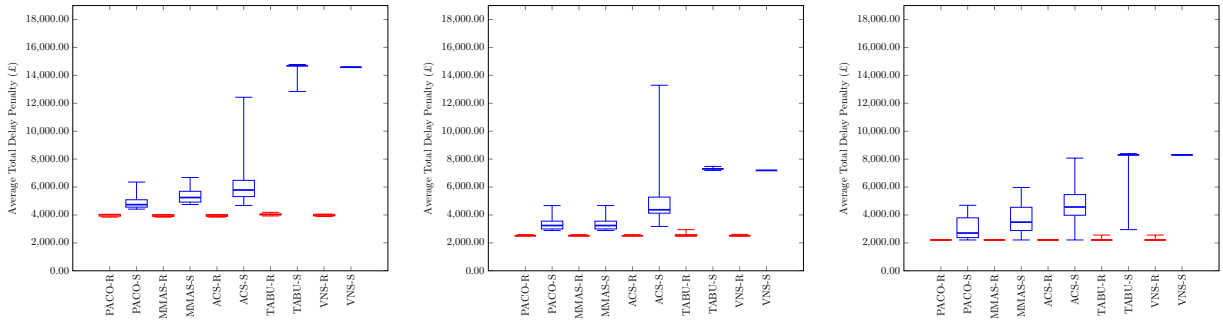
The most striking feature of the graphs is that, for all algorithms, just reordering the trains as they leave the station (blue boxes with S suffix) performs worse than just reallocating trains to new platforms (red boxes with R suffix). This may be because, as described in Section 6.2.2, reallocation may indirectly change a train's departure order from the station while resequencing can only re-order the trains on their original platform. In addition, for the resequencing sub-problem there is a large difference between the performance of the different algorithms. ACS-S appears to perform much worse than MMAS-S and PACO-S and, in the low magnitude, low-frequency dynamic scenarios it performs worse that TABU-S and VNS-S. In addition, the box plots for ACS are fairly large, suggesting that ACS produced a wide variation in solution quality for the resequencing problem and was less consistent in finding a good solution.

TABU-S and VNS-S perform much worse than the ACO algorithms on the high magnitude, high frequency delay scenarios, suggesting that when disruption is high the local search algorithms are not able to find a good solution.

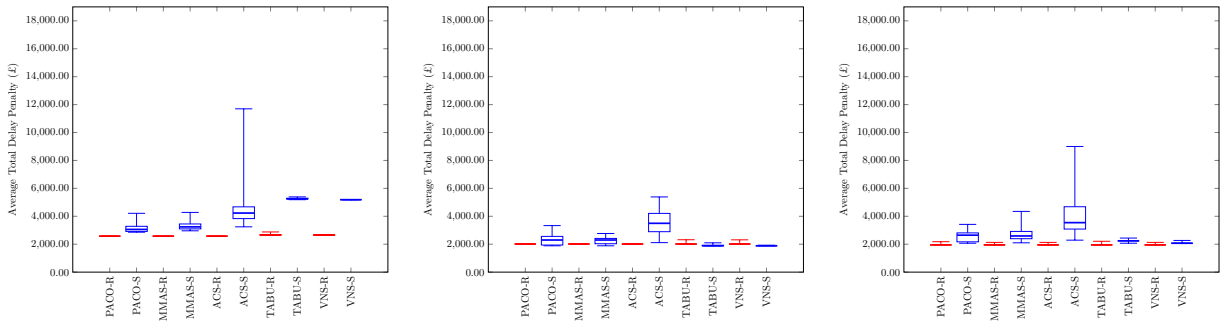
Results were tested for statistical significance using the Kruskal-Wallis test for multiple comparisons followed by the Wilcoxon rank-sum, non-parametric, pairwise test with Bonferroni correction at a 0.05 significance level. Table 6.7 shows a statistical analysis of the results for the reallocation sub-problem, while table 6.8 shows



m=30 f=10 m=30 f=20 m=30 f=30



m=20 f=10 m=20 f=20 m=20 f=30



m=10 f=30 m=10 f=20 m=10 f=30

Figure 6.12: Box plot comparison for total delay penalty averaged over all changes for each delay scenario

a statistical analysis of the results for the resequencing sub-problem. These tables show the results of comparing Algorithm1 $\Leftrightarrow$ Algorithm2, where the symbol '$s+$' indicates that Algorithm1 is significantly better than Algorithm2 while '$s-$' indi-

Table 6.7: Statistical analysis of average delay penalty for each dynamic scenario at 0.05 significance level for the Reallocation sub-problem

	$m = 30$			$m = 20$			$m = 10$		
$f =$	10	20	30	10	20	30	10	20	30
MMAS-R $\Leftrightarrow$ PACO-R	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
MMAS-R $\Leftrightarrow$ ACS-R	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
MMAS-R $\Leftrightarrow$ TABU-R	$s+$	$s+$	$\sim$	$s+$	$\sim$	$\sim$	$s+$	$\sim$	$\sim$
MMAS-R $\Leftrightarrow$ VNS-R	$\sim$	$s+$	$\sim$	$s+$	$\sim$	$\sim$	$s+$	$\sim$	$\sim$

Table 6.8: Statistical analysis of average delay penalty for each dynamic scenario at 0.05 significance level for the Resequencing sub-problem

	$m = 30$			$m = 20$			$m = 10$		
$f =$	10	20	30	10	20	30	10	20	30
PACO-S $\Leftrightarrow$ MMAS-S	$s+$	$\sim$	$s+$	$s+$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
PACO-S $\Leftrightarrow$ ACS-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
PACO-S $\Leftrightarrow$ TABU-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$\sim$
PACO-S $\Leftrightarrow$ VNS-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s-$

cates that Algorithm1 is significantly worse than Algorithm2, and the symbol '$\sim$' indicates no significant difference between the two algorithms.

For the reallocation sub-problem (see Table 6.7) there is no significant difference between the ACO algorithms in terms of average delay penalty for any of the delay scenarios. However, MMAS-R performs significantly better than TABU-R and VNS-R on the high frequency, high magnitude delay scenarios. This is possibly because TS and VNS have no inbuilt mechanism to cope with the dynamic scenario and are unable to carry information from before the change to the next change period.

With regards to the resequencing sub-problem (see Table 6.8), PACO-S significantly outperforms ACS-S on all scenarios and MMAS-S on the high magnitude and high frequency scenarios. It is an interesting question as to why PACO performs better than MMAS and ACS on the resequencing sub-problem but not on the reallocation problem. One of the main differences between these two problems is the use of a heuristic. In the reallocation sub-problem, a platform displacement heuristic is used to guide the ants, but in the resequencing sub-problem there is no suitable heuristic available therefore the ants are guided only by the pheromone

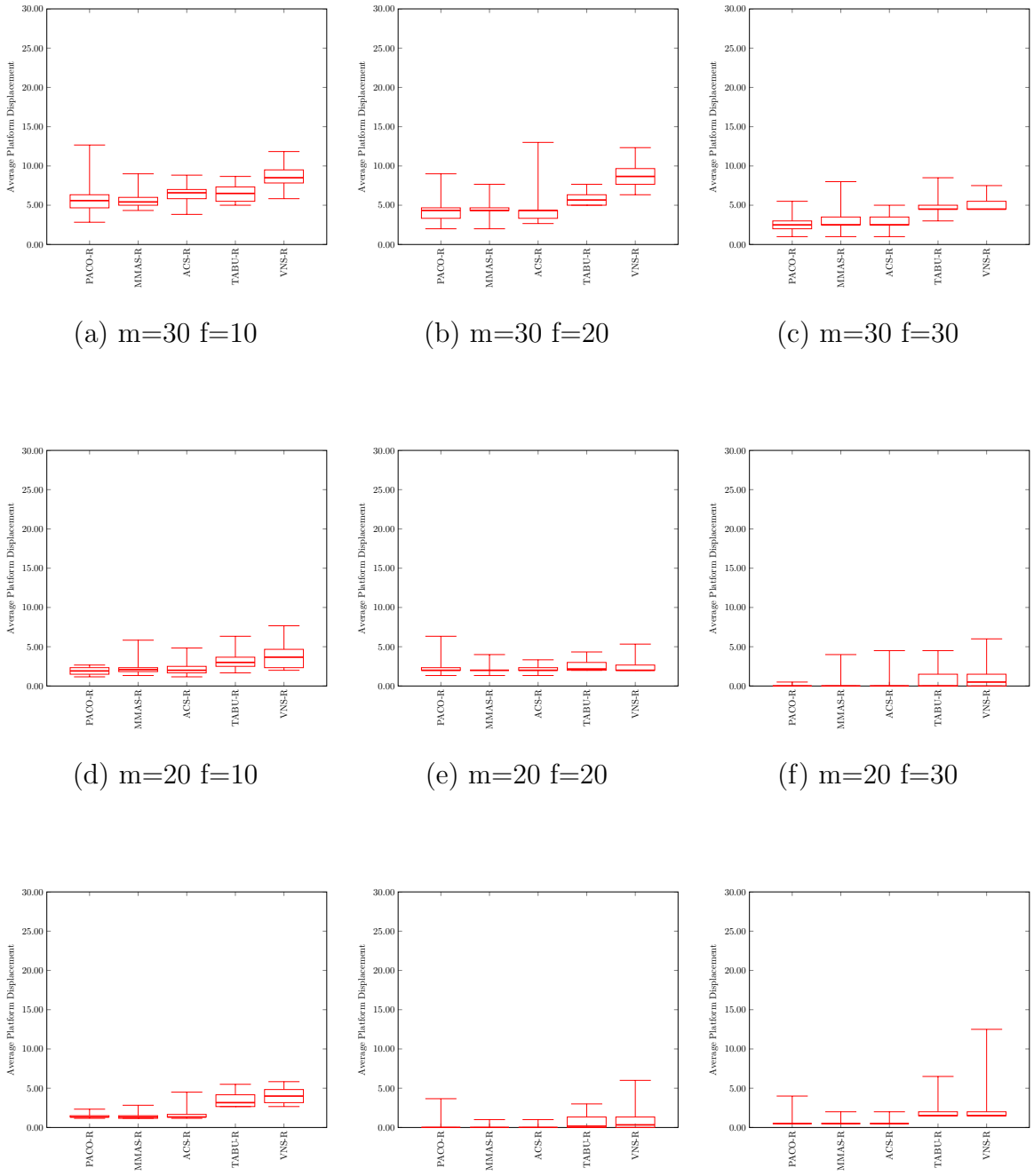
Table 6.9: Statistical analysis of average delay penalty for each dynamic scenario at 0.05 significance level comparing PACO-R and MMAS-R with PACO-S

	$m = 30$			$m = 20$			$m = 10$		
$f =$	10	20	30	10	20	30	10	20	30
PACO-R $\Leftrightarrow$ PACO-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$\sim$	$s+$
MMAS-R $\Leftrightarrow$ PACO-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$\sim$	$s+$

values. It may be that MMAS and ACS are particularly reliant on the use of a heuristic for their efficient operation and the absence of a heuristic is detrimental to their performance. However, another possible reason may be due to the way that the algorithms carry information from one change period to the next. In PACO the information is carried forward in a memory of repaired solutions, in MMAS and ACS it is carried in the pheromone trails with redundant trails being slowly removed over time by evaporation. It may be that, in line with the findings in Chapter 4, in the resequencing problem the stronger approach of clearing all pheromone trails and reinitialising them with the repaired solution in memory, as in PACO, may be more effective than retaining the pheromone trails and waiting for pheromone evaporation to remove the redundant information over time, as in MMAS and ACS.

In the resequencing sub-problem, both VNS and TS perform significantly better than PACO-S on the lowest magnitude and frequency changes. This suggest that in a scenario with only a small amount of disruption VNS and TS may perform better than ACO on this problem. Examination of the platform displacement box plots (see Fig. 6.13(h)) and 6.13(i)) shows that for these scenarios platform displacement is very low, suggesting that a good solution can be found without changing the trains original platforms. In this case simply resequencing using a local search based algorithm works better than using ACO.

A comparison of the best performing COLONY-R ACO algorithms with the best performing COLONY-S algorithm (see Table 6.9) shows that the reallocation colonies nearly always outperform the resequencing colonies in terms of average delay penalty. This confirms the results shown in the box plots (see Fig. 6.12) and is probably because resequencing alone has less flexibility to find good solutions than reallocation as it cannot change the trains platforms. While reallocation can change the platforms allocated to trains and also indirectly change the order that the trains leave the station (see Section 6.2.2).



(a) m=30 f=10 (b) m=30 f=20 (c) m=30 f=30

(d) m=20 f=10 (e) m=20 f=20 (f) m=20 f=30

(g) m=10 f=30 (h) m=10 f=20 (i) m=10 f=30

Figure 6.13: Comparison between platform displacement averaged over all changes for each delay scenario

Average Platform Displacement

With regards to the average platform displacement, Fig. 6.13 shows box plots for the comparison of platform displacement averaged over all changes for each algorithm.

Table 6.10: Statistical analysis of average platform displacement for each dynamic scenario at 0.05 significance level for the Reallocation sub-problem

	$m = 30$			$m = 20$			$m = 10$		
$f =$	10	20	30	10	20	30	10	20	30
MMAS-R $\Leftrightarrow$ PACO-R	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
MMAS-R $\Leftrightarrow$ ACS-R	$s+$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
MMAS-R $\Leftrightarrow$ TABU-R	$s+$	$s+$	$s+$	$s+$	$\sim$	$\sim$	$s+$	$s+$	$s+$
MMAS-R $\Leftrightarrow$ VNS-R	$s+$	$s+$	$s+$	$s+$	$\sim$	$s+$	$s+$	$s+$	$s+$

Only platform displacement for the reallocation sub-problem (red boxes and suffix R) are shown. This is because resequencing does not change the platforms allocated to the trains and therefore platform displacement will always be zero.

The box plots show that average platform displacement increases with higher magnitudes of delay and with more frequent delays. This suggests that more trains need to be reallocated in high frequency and high magnitude delay scenarios. In all cases the average platform displacement is worse for TABU-R and VNS-R. This is because, even though the same best-so-far replacement scheme is used for these algorithms, the fact that there is no heuristic means that there is less pressure to minimise platform displacement than for the ACO algorithms that use the best-so-far replacement scheme plus a heuristic to guide the ants.

Table 6.10 shows the statistical analysis of comparing each algorithm for average platform displacement for each delay scenario. The statistical analysis reveals no significant difference between MMAS-R and PACO-R however, ACS-R peforms significantly worse than MMAS-R on the high magnitude, high frequency delay scenario ($m = 30, f = 10$).

The statistical analysis reinforces the results observed in the box plots (see Fig. 6.13), MMAS performs significantly better than TABU-R and VNS-R in terms of average platform displacement for most of the delay scenarios. Again this illustrates the role the heuristic plays in minimising platform displacement.

6.7.2 Experimental Results using Two Colonies of Ants

The different outcomes for reallocating and resequencing the trains at the station after a delay suggests that combining the outcome of the two sub-problems may give an overall improved solution. In this set of experiments, two colonies of ants were combined to investigate if a multi-colony ant algorithm could improve the solutions

produced by one of the best single colony algorithms (MMAS-R).

The question arises as to which colonies should be combined. The previous results suggest that the best algorithm for the resequencing problem is PACO-S, therefore PACO-S is the algorithm implemented for the resequencing sub-problem.

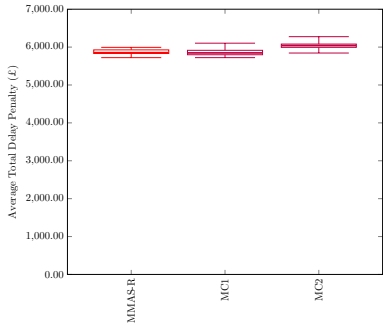
When considering reallocation, ACS-R performs significantly worse in terms of platform displacement, than MMAS-R, on the high magnitude, high frequency delay scenario ($m = 30, f = 10$). Therefore, ACS is not deemed a suitable candidate for the reallocation colony. PACO-R and MMAS-R give the same solution performance. Only one can be chosen for the reallocation sub-problem. In this case MMAS-R was chosen. However, in future work experiments will be carried out to also investigate the use of PACO-R as the algorithm for the reallocation sub-problem.

As detailed in Section 6.4 the algorithms are combined in two different ways. MC1 combines MMAS-R with PACO-S so that reallocation is performed and then resequencing. MC2 combines PACO-S with MMAS-R so that resequencing is performed before reallocation.

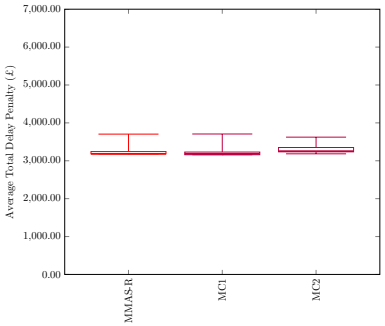
Average Total Delay Penalty

Fig. 6.14 gives box plots for the results of running these three algorithms for average total delay penalty. The box plots reveal that it is the magnitude of the change that has the biggest influence on the ability of the two-colony algorithm to improve the results found by the single colony algorithm. For the high magnitude scenarios ($m = 30$), MC1 provides no improvement while MC2 performs poorly compared to MC1 and MMAS-R. This suggests that changing the order the trains leave the station and then reallocating them to new platforms is not effective when the magnitude of the delay is high. For the medium magnitude scenarios ($m = 20$) there appears to be little difference between the three algorithms. In contrast, for the low magnitude delay scenarios ($m = 10$), both MC1 and MC2 show an improvement over MMAS-R. This improvement is particularly noticeable when the delay frequency is high, that is when there are large intervals between successive delays.

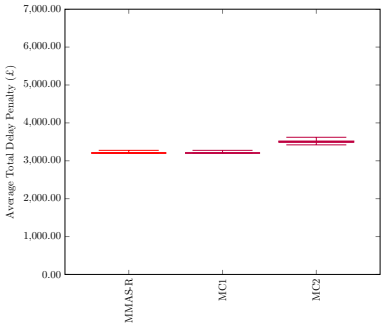
These results are confirmed in Table 6.11, which gives the statistical results of comparing the three algorithms. As can be seen MC2 performs significantly worse than MC1 and MMAS-R on all the high magnitude ($m = 30$) scenarios. In contrast, on some of the low and medium magnitude delay scenarios, MC2 gives an improvement in performance, as does MC1.



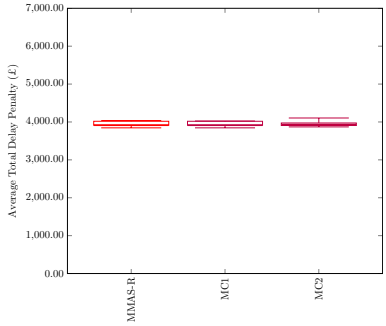
m=30 f=10



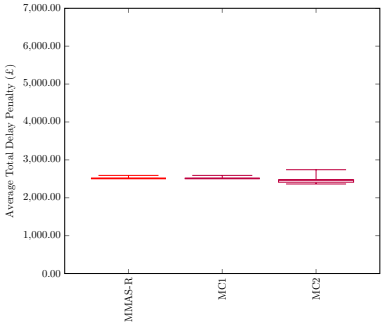
m=30 f=20



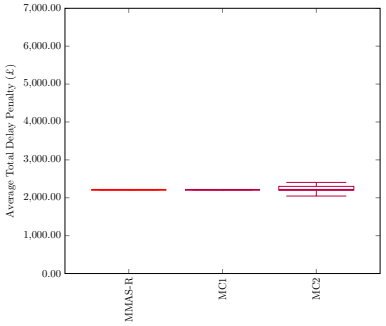
m=30 f=30



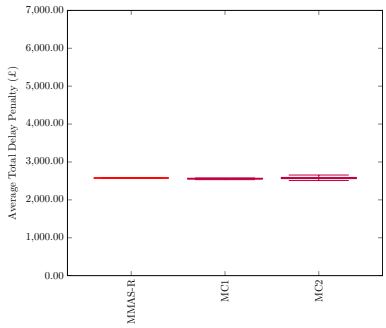
m=20 f=10



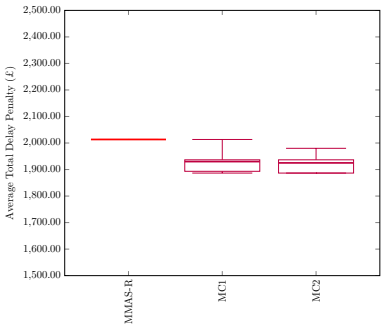
m=20 f=20



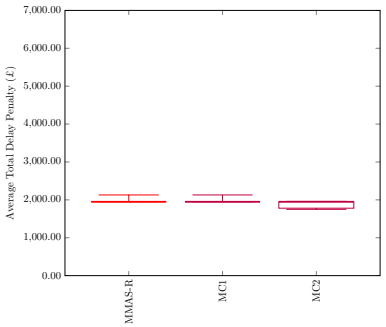
m=20 f=30



m=10 f=30



m=10 f=20



m=10 f=30

Figure 6.14: Box plot comparison for total delay penalty averaged over all changes for each delay scenario for the multi-colony algorithms

Table 6.11: Statistical analysis of average delay penalty for the multi-colony algorithms for each dynamic scenario at 0.05 significance level

	$m = 30$			$m = 20$			$m = 10$		
$f =$	10	20	30	10	20	30	10	20	30
MC1 $\Leftrightarrow$ MMAS-R	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$s+$	$s+$	$\sim$
MC2 $\Leftrightarrow$ MMAS-R	$s-$	$s-$	$s-$	$\sim$	$s+$	$\sim$	$\sim$	$s+$	$\sim$

Average Platform Displacement

Fig. 6.15 shows box plots for the average platform displacement for the multi-colony algorithms compared to MMAS-R. The plots show that platform displacement is greater with higher magnitude and higher frequency delay scenarios. In addition, MC2 performs worse than MMAS-R and MC1 in terms of reducing average platform displacement.

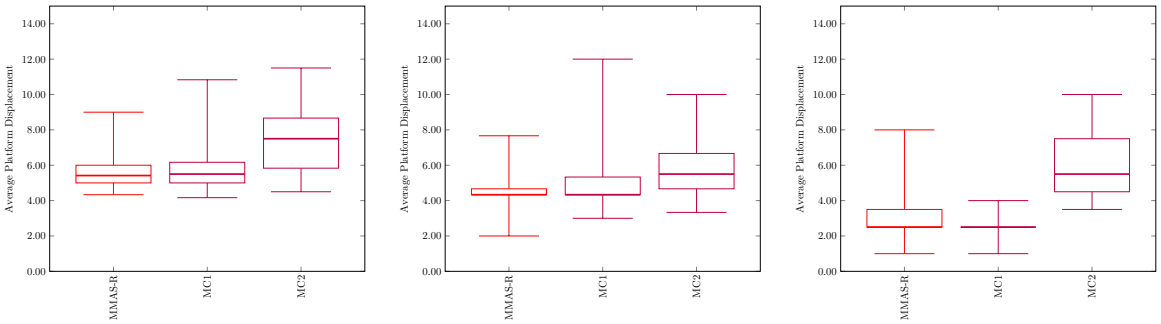
A statistical analysis was performed and the results are shown in Table 6.12, the results show that MC2 performs very poorly in terms of minimising platform displacement. Even when the difference is not significant MC2, performs worse than MMAS-R in terms of platform displacement, for example, for scenario $m = 20, f = 20$, the median displacement for MC2 is 2.333, whereas it is only 2 for MMAS-R. In contrast, there is no significant difference in average platform displacement between MC1 and MMAS-R. This suggests that the improvement shown by MC2 comes at the expense of increasing the platform displacement and that MC1 provides a better combination of the two colonies than MC2. To clarify, to minimise platform displacement in this problem, it is better to reallocate trains to new platforms and then reorder their departure from the station than to reorder the departure and then reallocate.

It can be seen that for the low magnitude, low and medium frequency, delay scenarios ($m = 10, f = 10$; $m = 10, f = 20$), MC1 significantly outperforms MMAS-R without any subsequent increase in platform displacement. This suggests that in low magnitude delay scenarios the two-colony MC1 algorithm provides an improvement in performance.

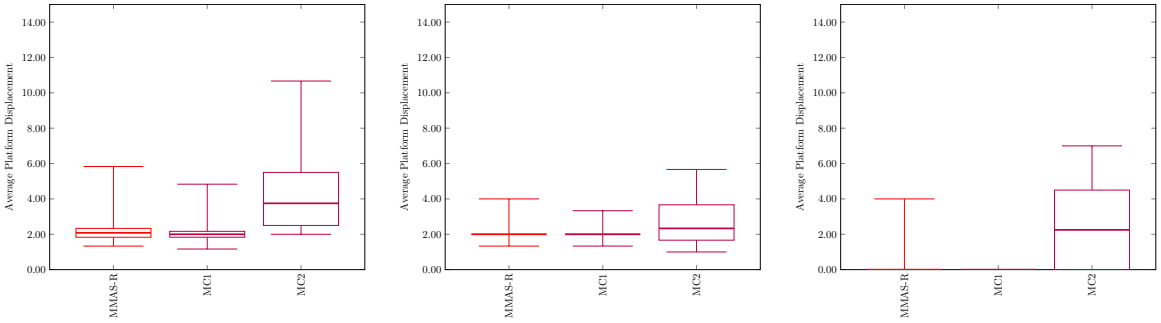
6.7.3 Comparison between MC1 and all Algorithms

Finally the performance of the best performing algorithm, MC1, is compared to running with all algorithms including NO-ALG (see Table. 6.13).

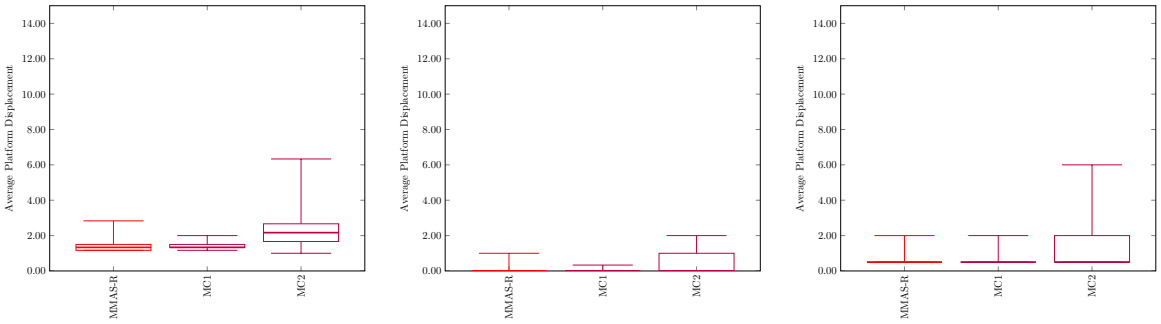
Results show that MC1 significantly outperforms NO-ALG on all delay scenarios.



(a) m=30 f=10 (b) m=30 f=20 (c) m=30 f=30



(d) m=20 f=10 (e) m=20 f=20 (f) m=20 f=30



(g) m=10 f=30 (h) m=10 f=20 (i) m=10 f=30

Figure 6.15: Box plot comparison for platform displacement averaged over all changes for each delay scenario for the multi-colony algorithms

Table 6.12: Statistical analysis of average platform displacement for the multi-colony algorithms for each dynamic scenario at 0.05 significance level

	$m = 30$			$m = 20$			$m = 10$		
$f =$	10	20	30	10	20	30	10	20	30
MC1 ⇔ MMAS-R	∼	∼	∼	∼	∼	∼	∼	∼	∼
MC2 ⇔ MMAS-R	$s-$	$s-$	$s-$	$s-$	∼	$s-$	$s-$	$s-$	∼

Table 6.13: Statistical analysis of average total delay penalty for MC1 compared with all other algorithms for each dynamic scenario at 0.05 significance level

	$m = 30$			$m = 20$			$m = 10$		
$f =$	10	20	30	10	20	30	10	20	30
MC1 ⇔ NO-ALG	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MC1 ⇔ PACO-R	∼	∼	∼	∼	∼	∼	$s+$	$s+$	∼
MC1 ⇔ MMAS-R	∼	∼	∼	∼	∼	∼	$s+$	$s+$	∼
MC1 ⇔ ACS-R	∼	∼	∼	∼	∼	∼	$s+$	$s+$	∼
MC1 ⇔ TABU-R	$s+$	$s+$	∼	$s+$	$s+$	∼	$s+$	$s+$	∼
MC1 ⇔ VNS-R	∼	$s+$	∼	$s+$	∼	∼	$s+$	$s+$	∼
MC1 ⇔ PACO-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MC1 ⇔ MMAS-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MC1 ⇔ ACS-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MC1 ⇔ TABU-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	∼	$s+$
MC1 ⇔ VNS-S	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$

In addition, it performs better than using PACO-R, MMAS-R and ACS-R on the low magnitude, medium to high frequency delay scenarios ($m = 10, f = 10$; $m = 10, f = 20$) and outperforms MMAS-S, PACO-S and ACS-S on all delay scenarios. It also outperforms TABU-R and VNS-R on almost all delay scenarios and outperforms TABU-S and VNS-S on all delay scenarios apart from $m = 10, f = 20$. This suggests that the two algorithms in MC1 work together to produce a better solution than reallocating or resequencing alone particularly in the low magnitude dynamic changes.

This seems counter-intuitive, however, it may be explained by the fact that, as mentioned in Section 6.2.2, the reallocation process provides some incidental

Table 6.14: Execution times for each algorithm in **seconds**, for delay scenario m=30, f=10.

Change	1	2	3	4	5	6
PACO-R	0.17	0.13	0.18	0.19	0.26	0.32
PACO-S	0.40	0.21	0.31	0.33	0.32	0.33
MMAS-R	0.19	0.13	0.20	0.21	0.30	0.34
MMAS-S	0.35	0.36	0.38	0.38	0.43	0.45
ACS-R	0.16	0.16	0.22	0.20	0.30	0.37
ACS-S	0.28	0.23	0.26	0.29	0.27	0.33
TABU-R	0.06	0.07	0.08	0.10	0.11	0.13
TABU-S	0.03	0.03	0.07	0.11	0.09	0.13
VNS-R	0.30	0.34	0.51	0.51	0.51	0.73
VNS-S	0.08	0.10	0.17	0.23	0.20	0.27
MC1	0.58	0.51	0.62	0.69	0.81	0.99
MC2	0.62	0.44	0.56	0.61	0.64	0.84

re-ordering of the trains just by changing the platforms the trains are allocated to. For the scenarios where MC1 outperforms MMAS-R, there is little platform displacement (see Fig. 6.15.h), which indicates very few trains have been allocated to new platforms. Therefore, the algorithm does not benefit from the station exit re-ordering inherent within the reallocation process and there is room for improvement by resequencing.

6.7.4 Time Analysis

The experiments were run on a 2.9GHz Intel Xeon E5-2666 v3 (Haswell) processor. Table 6.14 shows the average execution for all of the algorithms on the most disrupted scenario ($m = 30, f = 10$), while table 6.15 shows the average execution time on the least disrupted delay scenario ($m = 10, f = 30$). It is apparent that all algorithms can find a solution in a very short computation time even for the most complex high magnitude, high frequency scenarios. For example, MMAS-R took an average of 19 s for change 1 and 34 s for change 6 for the highly disrupted delay scenario ($m = 30, f = 10$) and 8 s for change 1 and 11 s for change 2 for the smallest disrupted delay scenario ($m = 10, f = 30$). The execution time increases with each

Table 6.15: Execution times for each algorithm in **seconds**, for delay scenario m=10, f=30.

Change	0	1
PACO-R	0.07	0.09
PACO-S	0.20	0.17
MMAS-R	0.08	0.11
MMAS-S	0.19	0.21
ACS-R	0.08	0.11
ACS-S	0.16	0.19
TABU-R	0.03	0.04
TABU-S	0.02	0.03
VNS-R	0.15	0.22
VNS-S	0.15	0.22
MC1	0.27	0.34
MC2	0.28	0.30

successive change as with each additional disruption more trains will be affected increasing the time taken to detect and resolve the conflict.

These values are within the maximum rescheduling time of 180 s suggested by the French operating company SNCF [58] and within the 5 min maximum rescheduling interval suggested by Meng and Zhou [67]. The two-colony algorithms, MC1 and MC2 have an unsurprisingly longer execution time. For example, MC1 has an computation time of 58s for change 1 and 99s for change 6 on scenario $m = 30, f = 10$. However, this is still within the acceptable range suggested by [58] and [67].

6.8 Summary

Rescheduling trains at stations after a delay is a complex task, made more complicated by the fact that it can be a dynamic problem that changes over time as more delayed trains arrive at the station. The DSRP can be considered to be two separate sub-problems. The first problem is to decide the platform to allocate to the delayed train; the second is to decide the order that the trains should leave the station. Solving these sub-problems can allow trains to overtake other trains at the station. In this work two colonies of ants have been used to address each of these

sub-problems. Investigations have been carried out to examine the effectiveness of using each colony separately and of combining them into a multi-colony algorithm.

In order to investigate the DSRP, a model was created from real-world train schedule data for Leicester station, a busy UK railway station. The model allows the impact of the local decisions made at the station to be projected into the future by accessing the impact the decisions have on a train's ongoing journey.

To evaluate the effectiveness of using ACO algorithms the results were compared with those obtained using TS, VNS and running with no platform reallocation or resequencing (NO-ALG). Results showed that the best performing ACO algorithms were MMAS and PACO and that they outperformed VNS and TS on the majority of the delay scenarios. It addition, it was observed that simply changing the order of the trains as they leave the station is not sufficient to obtain a good outcome. In contrast, just reallocating the trains to new platforms performs well in almost all cases. This is most likely because the reallocation process also performs some reordering of train departures, as explained in Section 6.2.2. Combining the reallocation and resequencing colonies, to form a multi-colony algorithm, did not change the performance on the high and medium magnitude delay scenarios but gave an improved performance on the low magnitude delay scenarios. However, this was only the case if reallocation was performed before resequencing.

The use of a platform displacement heuristic combined with a novel best-so-far ant replacement scheme worked well to give the ants the intelligence to minimise unnecessary platform changes. However, from the results it seems that a decrease in delay often results in an increase in platform displacement, especially for MC2 (see Section. 6.7.2). This suggest that minimising platform displacement and minimising delay may be conflicting objectives and future work will consider applying a dynamic, multi-objective ACO algorithm to this problem.

The algorithm executed in an acceptable time frame. For the single colony algorithm where the delay scenario had a small amount of disruption a solution could be found in an average of 19 sec for change 1 and 34 sec for change 6. The two-colony algorithm MC1 took longer to find a solution, which suggests that repeatedly iterating between the two colonies may take too long to be used in a real-world rescheduling scenario, however, this is also an area for future investigation.

Another possible avenue for future work is that of investigating the outcome of allowing the reallocation process to change the dwell time of the train. In this current investigation, dwell time can be changed by resequencing but not by reallocation. In some cases, dwell time may be large enough to absorb some of the train delays and reducing it may provide further improvement to the reallocation algorithm.

The fact that VNS performs better than MC1 on the delay scenario $m = 10, f = 20$ suggests that there are some delay scenarios that ACO does not perform as well on. An investigation of how to improve the performance of ACO on these scenarios warrants future work. As VNS is based on a local search, this suggests that the addition of a local search to the MC1 algorithm may improve it even further.

It is recognised that there are some limitations with the model, mainly as a result of the paucity of information available in the schedule feed. However, the model works as a 'proof-of-principle' and any additional information that could be obtained in the future could be used to refine it. An advantage of the model is that it does not rely on knowing the exact position of each train at every moment in time, via GPS sensors or similar, to work out the impact of the local station decisions on the trains' ongoing journeys. Any new information about the trains' positions could be used to update the train details before running for the next dynamic change period. This would allow new solutions to be made based on the current train positions.

The motivation for this work is to contribute towards the development of algorithms that could be implemented within a computer-based dispatching system to support the railway controller in solving schedule conflicts after a perturbation. The next step is to model a more complex station, such as Nottingham station, in order to apply the same algorithms. It is possible that in a busier station the two-colony algorithm may provide even more improvements. However, the creation of a model for Nottingham station would require more information than is available in the schedule feed. It is hoped that the results shown in the current work will encourage the provision of such information and will allow the work to be expanded.

Chapter 7

Conclusion

Delays are a common occurrence on the British railway network. Minimising the impact of a delay on the railway system is an important concern of railway operators and the effectiveness of a rescheduling solution has an impact both on railway passengers and train operating companies. However, railway rescheduling is a difficult problem, made more complicated by the fact it can be both dynamic and multi-objective. It is dynamic because the railway system is in a constant state of movement, while trains are in the process of being rescheduled, more delays may occur or trains with higher priority may arrive changing the nature of the problem over time. The problem is multi-objective because a train dispatcher may need to simultaneously minimise several conflicting consequences of the perturbation, such as delay, timetable deviation, energy consumption and missed connections. The conflicting nature of these objectives means that increasing the quality of one objective may have a detrimental effect on the quality of another.

Dynamic railway rescheduling problems are rarely addressed in the literature. Most train rescheduling problems are regarded as static in that all delays are known about in advance and it is assumed that no further unforeseen incidents occur while the original delay is being resolved. In this case, if a further incident occurs the algorithm would have to be restarted again from scratch with the loss of potentially useful information from before the change that could have been used in the new environment. There are a growing number of researchers that recognise a need for such work [5, 3, 7, 6] and one of the aims of this thesis was to take a step towards the investigation and resolution of such dynamic railway rescheduling problems.

Multi-objective railway rescheduling problems are more commonly considered but in most cases the objectives are weighted and combined to create a single objective and the problem is solved as a single objective problem. The disadvantage of this method is that the weights have to be determined in advance using domain

knowledge and it assumes that the relative importance of each objective does not change over time. A more flexible approach is to produce a set of trade-off solutions to provide the decision maker with a choice of solutions. This will allow them to make a decision as to which solution best matches their requirements at a particular moment in time.

The goal of this thesis was to address a gap in railway rescheduling research, that of rescheduling trains in dynamic environments for both single-objective and multi-objective problems. This is seen as an important step towards the development of algorithms for computerised dispatching systems as many real-world scheduling problems may be both multi-objective and dynamic.

To achieve this goal three different dynamic multi-objective problems were created. The first was a single objective dynamic junction rescheduling problem based on the Stenson junction on the British railway network. The second was an extension of the Stenson junction problem, to make it a dynamic multi-objective problem, by the addition of another objective, minimising energy consumption. The third was based on a model created from Network Rail's schedule data feed for Leicester station. This model considered the effect that the local rescheduling decisions made at the station had on the wider network.

A summary of the findings for each of those problems is given below:

7.1 Railway Junction Rescheduling in Dynamic Environments

This problem is referred to as the dynamic railway junction rescheduling problem (DRJRP) in this work. In the DRJRP, the environmental change is a result of the arrival of new timetabled trains while the original trains are waiting to be rescheduled at the junction. In the extended DRJRP multiple unrelated delays occur over the time period of the investigation. The extra disruptions are caused by trains being delayed at the stations that feed into the railway junction.

The results showed that:

- P-ACO outperforms FCFS on the high to medium magnitude and high to medium frequency changes.
- When considering multiple delays, with a change magnitude of eight trains, FCFS was outperformed by all the P-ACO algorithms.
- In this problem, ACO algorithms with a memory (P-ACO) were found to

cope with dynamic changes better than an ACO algorithm that uses only pheromone evaporation (MMAS) to remove redundant pheromone trails.

- In P-ACO, when the ants in memory cannot be modified to make them feasible in the new environment, replacing the memory with elite immigrants after a change works effectively.
- Random immigrants were found to be unsuitable to replace the ants in memory when changes were of a high magnitude and a high frequency. The larger search space appears to demand the knowledge carried over from previous environments.
- Adding the ability to sequence trains at the stations was not beneficial to this set of dynamic problems. This may be because the extra decisions that the ants have to make increases the size of the search space and the ants struggle to explore it adequately. This suggestion is supported by the fact that when the search space is relatively small in the low frequency scenarios, algorithms that sequence the trains at the stations (EI-PACO, HI-PACO and MMAS) perform slightly better than the algorithm without station sequencing (NSS-PACO).

7.2 Multi-objective Railway Rescheduling in Dynamic Environments

In this work the DRJRP was extended by the addition of a second objective, that of minimising energy consumption. It now becomes the dynamic multi-objective junction rescheduling problem (DM-RJRP). Several different multi-objective ACO (MOACO) algorithms were applied to the problem; based on a population based ACO (P-ACO) [94], and on the *MAX-MIN* Ant System (MMAS) [37]. Each algorithm uses a different method of dealing with dynamic changes. The best ACO algorithm was compared with NSGA-II [119], a 'state-of-the-art' multi-objective algorithm, and FCFS. An additional goal of this work was to attempt to identify the features of an ACO algorithm that makes it suitable for coping with both the dynamic as well as multi-objective nature of this problem.

The results showed that:

- All the ACO algorithms were able to find a POS of solutions for the DM-RJRP. However, the algorithm based on P-ACO performed better than the algorithms based on MMAS.

- The performance of multi-objective MMAS can be improved, on this problem, by retaining the non-dominated archive between changes.
- On scenarios with large and frequent changes, multi-objective MMAS also benefits from retaining the pheromone trails between changes.
- The best performing algoirithm DM-PACO-R outperformed NSGA-II and FCFS.

7.3 Platform Reallocation And Resequencing in Dynamic Environments

The above work is focused on a small area of the railway network, the aim with this work was to take into account the effect that changes made in a local area have on the global behaviour of the network. To investigate this a macroscopic model of the railway was created based on Network Rail's train schedule feed [30]. The model details both the movement of trains through the station and the movement of all trains at each of the timing points on the trains' routes. This allows the long-term consequences of the reallocation decisions to be determined. The radius considered was 50 miles (80.47 km) of the station which covered approximately 225 timing points. The use of real schedule data makes the results applicable to a real-world dispatching system as the same data is available to dispatchers when rescheduling trains.

The problem of rescheduling trains at stations after a delay can be divided into two sub-problems. The first problem, the reallocation sub-problem, is to decide the platform to allocate to the delayed train; the second, the resequencing sub-problem, is to decide the order that the trains should leave the station. Solving these sub-problems can allow trains to overtake other trains at the station. This is important because often the limitations of the railway infrastructure means that overtaking is only possible at stations [21]. In this work two colonies of ants were used to address each of these sub-problems. Investigations were carried out to examine the effectiveness of using each colony separately and of combining them into a multi-colony algorithm. The algorithms performance was evaluated by comparing with TS, VNS and running with no platform reallocation or rescheduling (NO-ALG)

The results showed that:

- P-ACO, MMAS and ACS all performed similarly on the reallocation sub-problem. However, ACS and MMAS were outperformed by P-ACO on the resequencing sub-problem.

- P-ACO and MMAS outperformed VNS and TS on the majority of the delay scenarios.
- Just resequencing the trains as they leave the station did not perform as well as reallocating the trains to new platforms. This is most likely because the reallocation process also performs some reordering of train departures, as explained in Section 6.2.2.
- Combining the reallocation and rescheduling colonies, to form a multi-colony algorithm, did not affect the performance on the high and medium magnitude delay scenarios but gave an improved performance on the low magnitude delay scenarios. However, this was only the case if reallocation was performed before resequencing.
- The use of a platform displacement heuristic combined with a novel best-so-far ant replacement scheme worked to give the ants the intelligence to minimise unnecessary platform changes.
- Both the single-colony and the two-colony algorithms executed in an acceptable time frame to be used in a real-world delay scenario. For example, the one-colony algorithm, MMAS-R, took an average of 34 s for the most perturbed delay scenario and 11 s for the least perturbed delay scenario. While the two-colony algorithm, MC1, had an average execution time of 99 s for the most perturbed delay scenario and less than a minute (58 s) for the least perturbed delay scenario.

7.4 Results Summary

The results of this work show that ACO algorithms have a part to play in dynamic and dynamic, multi-objective, rescheduling problems.

In addition, for the Leicester station problem (DSRP), using a macroscopic model, the algorithm executes in an acceptable time frame to be implemented in a real-time dispatching system. The fact that the model makes use of the same timetable data that is available to railway dispatchers ensures that the problem is applicable to real-time implementation.

Although the microscopic Stenson junction model takes much longer to execute, the process could be speeded up by terminating the algorithm if there has been no change in the best-so-far solution in a set number of iterations.

It is also apparent, from the results, that the magnitude and frequency of change has an affect on the performance of the algorithms. Generally low magnitude, low frequency changes can be solved with simple heuristics, such as FCFS. However, high magnitude, high frequency changes benefit from the use of the ACO algorithms. In the DM-RJRP it was not only the magnitude and frequency of the change that affected the performance of the algorithm, but also the interaction between the objectives.

The results also show that in dynamic railway rescheduling problems, retaining information between changes is beneficial to the performance of the algorithms, especially when the changes are of a high magnitude and frequency. For example, in Chapter 4 it was shown that using random immigrants to replace the memory after a change (RI-PACO) impairs the performance of the algorithm on the high magnitude, high frequency change scenario. Using random immigrants to initialise the pheromone trails after a change is effectively a restart of the algorithm. This shows the importance of retaining the information between changes especially when the magnitude and frequency of change is high.

Similar results were found in Chapter 5, where it was found that retaining the non-dominated archive of solutions between changes improved the performance of the MMAS algorithm when the changes were of a high frequency and of a medium to high magnitude. Retaining the non-dominated archive between changes can be thought of as keeping a memory of the solutions found before and this information was found to be useful for finding good solutions in the next dynamic change.

The reason why retaining the information between changes is important in the high magnitude and high frequency delay scenarios is because when many trains are added in short intervals, few trains will have had the opportunity to pass through the junction before the next set of trains arrives. This results in a large number of trains in the system and a correspondingly large search space for the ants to navigate. The large search space may make it difficult for the ants to find good new solutions especially as the good solutions may now have become localised in one area of the search space due to time-linked nature of the problem. Retaining the information from the previous change period helps to guide the algorithm in their search for better solutions.

7.4.1 Comparison with Existing Railway Rescheduling Work

It is difficult to make a direct comparison of the results of this research with previous research because of the range of different models and problem scenarios employed.

There is, unfortunately, at the present time no single benchmark problem that can be used by all researchers to test the outcomes of different approaches.

However, in terms of approaches to modelling the problem, the approach in this thesis may have an advantage in dynamic delay scenarios over the popular alternative graph method used by [42, 18, 54, 56, 55, 19]. Alternative graphs require a preprocessing step in which the graph is created using the predefined routes of each train to create the fixed and alternative arcs [62]. In a dynamic delay situation this would have to be recalculated every time a change occurs, as new trains will have arrived, some trains will be lost and arrival and departure times may have changed due to a previous delay. In contrast, the algorithms in this thesis use a snapshot of the network at the time of change based on the best solution found before the change. This is easily updated using real-time train information and only the structure of the graph given to the algorithms needs to be updated after a change. The graph given to the algorithms does not require train times or exact positions of trains which reduces the amount of 'rebuilding' that has to take place after a change in order to solve the new problem.

A further advantage of the macroscopic approach taken in the DSRP is that it does not rely on knowing the exact position of each train at every moment in time, via GPS sensors or similar, to work out the impact of the local station decisions on the trains' ongoing journeys. Any new information about the trains' positions could be used to update the train details before running for the next dynamic change period. This would allow new solutions to be made based on the current train positions. In contrast, an alternative graph approach assumes the use of GPS sensors on trains to provide real time position data to the dispatching control centre [55].

7.5 Unique Contribution

The unique contributions of this work to the study of railway rescheduling include:

7.5.1 Dynamic Rescheduling Problems

- The creation of a benchmark problem and simulator to investigate dynamic railway rescheduling problems. Such problems are rarely considered in current railway rescheduling research.

- A contribution to the field of understanding of how ACO algorithms can be applied to dynamic railway rescheduling problems. Very few researchers have applied ACO algorithms to railway rescheduling problems. Those that have,

Fan *et al.* [2] and Samà *et al.* [23], applied it only to static rescheduling problems.

- A contribution to the field of understanding of the effect the characteristics of the delay, in terms of magnitude and frequency, have on the ability of the algorithms to solve dynamic rescheduling problems.

7.5.2 Dynamic Multi-objective Rescheduling Problems

- The creation of a benchmark problem and simulator to investigate dynamic multi-objective railway rescheduling problems.
- The investigation of a railway rescheduling problem that is both dynamic and multi-objective. As previous works consider only static or multi-objective problems, they fail to take into account the dynamic and multi-objective nature of railway scheduling problems. The investigation of such a problem is a new contribution to railway rescheduling.
- A contribution to the field of understanding of how ACO algorithms can be applied to railway rescheduling DMOPs and an attempt to identify both the features of the algorithms necessary for good performance on this DMOP and also the effect of the frequency and magnitude of change on each algorithm's performance.

7.5.3 Dynamic Platform Reallocation and Rescheduling

- The creation of a benchmark problem for a dynamic station rescheduling problem based on Network Rail's schedule feed [30].
- A contribution to the field of understanding of how ACO algorithms can be applied to the field of dynamic railway rescheduling, specifically dynamic platform reallocation and resequencing.
- The creation of a unique framework that combines two colonies of ants, one that reallocates trains to platforms after a delay and the other that determines the order the trains leave the station.

7.6 Limitations

Although, the models attempt to be as realistic as possible, the fact that they are simulations means that they are not exact representations of real-world rescheduling

problems.

7.6.1 Limitations of the Stenson Junction Simulator

To make the Stenson junction simulator as realistic as possible, it is based on a real section of the British railway network, and the mechanics of railway operation, such as interlocking and automatic fixed block technology have been included in the simulation. In addition, the model uses power and resistance data based on RailSys data, which is used by Network Rail as a simulation tool [109].

However, at this present time, Network Rail are unable to provide the data necessary to investigate dynamic and dynamic multi-objective railway rescheduling problems therefore delays have been simulated in order to investigate such problems.

Other limitations of this simulator is that it considers only a small area of the British railway network and considers only the local impact of the rescheduling decisions. As this is a microscopic model, increasing the area covered would increase the execution time of the algorithm. A further limitation of the model is that it considers only flat sections of track with no gradients. The addition of gradients into the problem would affect the energy consumption of the trains and may impact the shape of the POF obtained by the algorithms.

7.6.2 Limitations of the Leicester Station Model

The development of the Leicester Station model was an attempt to address some of the limitations of the Stenson junction simulator, that of only considering a small area of the British railway network and not taking into account the effect of local decisions on the global performance of the network.

It is recognised that due to the lack of accurate data about the movement of trains on the network the model has limitations. For example, assumptions had to be made about which terminating trains became originating trains. In addition, due to lack of information about the maximum speed of each train the running time of each train on each track section is the running time extracted from the timetable and is fixed. A further limitation is that the problem does not consider train connections, again this is due to the fact that such data is missing from the train schedule feed. In addition, the problem does not take into account crew rescheduling or all rolling stock connections. Although the DSRP takes into account terminating trains that become originating trains, other rolling stock issues such as trains being connected together to form new services are not considered. This is due to the lack of information about such activities in the train schedule feed.

These limitations could be resolved by the provision of more descriptive data from Network Rail and even though the model itself lacks some details it still works to provide a prototype model to investigate the application of ACO to dynamic station rescheduling problems.

7.7 Wider Impact

Although this work focused on train rescheduling it does have the potential for a wider impact based on its application to other real-world dynamic multi-objective transport rescheduling problems. Public transport systems, such as bus services, are also subject to unexpected events, such as traffic jams, breakdowns etc., that can cause delays. Again, unrelated delays may successively occur making the problem a dynamic one that changes over time. In addition, there may be multiple objectives such as reducing disruption and minimising operation costs.

A further area of application is in the routing of autonomous vehicles. This is a dynamic problem as vehicles may need to dynamically reroute to avoid unexpected obstacles or traffic jams. This problem may also be multi-objective as the intention may be to optimise several conflicting objectives simultaneously, such as distance travelled, speed and energy consumption.

Another area of application is in industrial scheduling and production planning. In this case problems may be both multi-objective and dynamic. The objectives may include minimising makespan, minimising processing costs, minimising energy consumption etc., while the dynamism may be a feature of changing orders, fluctuating resource availability and machine breakdowns.

The work is also applicable to dynamic multi-objective task scheduling problems in cloud computing where the objectives may be to minimise CPU memory usage, execution time and execution cost and the dynamism is a result of changing demand over time.

Another real-world area of application is in the financial world. The optimisation of a portfolio of shares can be a multi-objective problem as the goals may be to minimise risk and maximise income while the dynamics are a result of the value of stocks and shares changing over time.

7.8 Future Work

There are a number of avenues of future work waiting to be explored, particularly for the DSRP. The first is that of maintaining station connections in the Leicester station

model. With more information about connections, minimising broken connections could be added as a second objective to the problem, making the Leicester station problem both a dynamic and a multi-objective problem.

In addition, although the use of a platform displacement heuristic combined with a novel best-so-far ant replacement scheme worked to give the ants the intelligence to minimise unnecessary platform changes. The results indicate that a decrease in delay penalty often results in an increase in platform displacement (see Section 6.7.2). This suggest that minimising platform displacement and minimising delay may be conflicting objectives and future work will consider applying a dynamic, multi-objective ACO algorithm to this problem.

Another possible avenue for future work is that of investigating the effect of allowing the reallocation process within the station problem to change the dwell time of the train. At the moment it is fixed to that of the time table, but in some cases the dwell may be large enough to absorb some of the train delays. In addition, the fact that VNS performs better than MC1 on one of the low magnitude delay scenarios suggests that there are some problems that ACO does not perform as well on. An investigation of how to improve the performance of ACO on these problems warrants future work. As VNS is based on a local search, this suggests that the addition of a local search to the MC1 algorithm may improve it even further.

A further very interesting way this problem could be transformed into a multi-objective problem is considering the problem for two different stations, for example Leicester and Nottingham. It is possible that minimising delay at one station may increase the delay at another station. For example, consider three trains, A, B and C. Train A is the delayed train. Train B passes through Leicester and Nottingham. Train C passes through Leicester but not Nottingham. Delaying Train B at Leicester station to accommodate Train A may result in less delay at Leicester. However, it may be better to delay Train C, from the point of view of Nottingham station, as it does not pass through Nottingham and its delay will have no impact on Nottingham. This suggests conflicting objectives and an interesting dynamic multi-objective railway rescheduling problem.

It is possible that in a real world dynamic delay rescheduling situation the dispatcher may want the optimum solutions in the new change period to be as close as possible to the optimum solutions in the previous change period. This could reduce the need to constantly update the instructions given to train drivers and crew. This is an area worth investigating and would involve identifying an optimal solution after the change that solves the problem but with minimum distance from the solution before the change.

A further area for future investigation is that of the time-linked nature of dynamic problems in railway rescheduling. A solution chosen in the previous change period will have been partially implemented before the next change in the environment which restricts the choice of future actions for the algorithm. It is important to explore the steps that can be taken to minimise the effect of time-linkage in dynamic railway rescheduling problems.

This is quite a long list of future work, but it is a good sign as it shows the rich vein of future research work that this thesis has opened up.

7.9 Conclusion

The motivation for this work has been to create a step towards a system that could be implemented within a computer-based dispatching system to support the railway controller in solving schedule conflicts after a perturbation. As the railway rescheduling problem may be both dynamic and multi-objective, addressing these aspects of the railway rescheduling problem ensures the algorithms will be more applicable to real-world problems.

In this work ACO has been found to be effective in finding good solutions in dynamic and dynamic multi-objective railway rescheduling problems. It has demonstrated the potential of ACO algorithms for solving such problems and has opened up an interesting avenue of research which should be of great interest to Network Rail. What is required from Network Rail now, is the realisation that the collection of data from dynamic delay scenarios will contribute towards the development of algorithms that can be integrated into computerised systems to help resolve disruptions after a delay in a dynamic environment with multiple objectives. The provision of such data will greatly improve the speed of research and will facilitate the expansion of the work presented in this thesis.

Bibliography

[1] J. Eaton and S. Yang, "Dynamic railway junction rescheduling using population based ant colony optimisation," in *Computational Intelligence (UKCI), 2014 UK Workshop on*, pp. 1–8, Sept 2014.

[2] B. Fan, C. Roberts, and P. Weston, "A comparison of algorithms for minimising delay costs in disturbed railway traffic scenarios," *Journal of Rail Transport Planning & Management*, vol. 2, pp. 23–33, 2012.

[3] N. Rail, "Network rail annual report and accounts 2016," 2016. [Accessed on 10th September 2016].

[4] ON-TIME, "Functional and technical requirements specification for large scale perturbation management," 2013. [Accessed on 10th January 2017].

[5] L. Meng and X. Zhou, "Robust single-track train dispatching model under a dynamic and stochastic environment: a scenario-based rolling horizon solution approach," *Transportation Research Part B: Methodological*, vol. 45, no. 7, pp. 1080–1102, 2011.

[6] F. Corman and L. Meng, "A review of online dynamic models and algorithms for railway traffic management," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 3, pp. 1274–1284, 2015.

[7] V. Cacchiani, M. Huisman, D.and Kidd, L. Kroon, P. Toth, and J. Veelenturf, L.and Wagenaar, "An overview of recovery models and algorithms for real-time railway rescheduling," *Transportation Research Part B: Methodological*, vol. 63, pp. 15–37, 2014.

[8] T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, no. 0, pp. 1 – 24, 2012.

[9] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Massachusetts, USA: Kluwer Academic Publishers, 2002.

[10] M. Samà, F. Corman, and D. Pacciarelli, "A variable neighbourhood search for fast train scheduling and routing during disturbed railway traffic situations," *Computers & Operations Research*, vol. 78, pp. 480–499, 2017.

[11] T. Nguyen and X. Yao, "Dynamic time-linkage problems revisited," in *Applications of Evolutionary Computing* (M. Giacobini, A. Brabazon, S. Cagnoni, G. Di Caro, A. Ekárt, A. Esparcia-Alcázar, M. Farooq, A. Fink, and P. Machado, eds.), vol. 5484 of *Lecture Notes in Computer Science*, pp. 735–744, Springer Berlin Heidelberg, 2009.

[12] S. Wegele and E. Schnieder, "Dispatching of train operations using genetic algorithms," in *Proceedings of the 9th International Conference on Computer-Aided Scheduling of Public Transport (CASPT), San Diego, USA*, Citeseer, 2004.

[13] S.-C. Chang and Y.-C. Chung, "From timetabling to train regulation–a new train operation model," *Information and Software Technology*, vol. 47, no. 9, pp. 575–585, 2005.

[14] N. Zhao, C. Roberts, S. Hillmansen, and G. Nicholson, "A multiple train trajectory optimization to minimize energy consumption and delay," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2363–2372, 2015.

[15] J. Pochet, S. Baro, and G. Sandou, "Supervision and rescheduling of a mixed cbtc traffic on a suburban railway line," in *Intelligent Rail Transportation (ICIRT), 2016 IEEE International Conference on*, pp. 32–38, IEEE, 2016.

[16] F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo, "Bi-objective conflict detection and resolution in railway traffic management," *Transportation Research Part C: Emerging Technologies*, vol. 20, no. 1, pp. 79 – 94, 2012.

[17] A. Fernández-Rodríguez, A. Fernández-Cardador, and A. Cucala, "Energy efficiency in high speed railway traffic operation: a real-time ecodriving algorithm," in *Environment and Electrical Engineering (EEEIC), 2015 IEEE 15th International Conference on*, pp. 325–330, IEEE, 2015.

[18] F. Corman, R. Goverde, and A. D'Ariano, "Rescheduling dense train traffic over complex station interlocking areas," in *Robust and Online Large-Scale Optimization*, pp. 369–386, Springer, 2009.

[19] P. Kecman, F. Corman, A. D'Ariano, and R. Goverde, "Rescheduling models for railway traffic management in large-scale networks," *Public Transport*, vol. 5, no. 1-2, pp. 95–123, 2013.

[20] I. Hansen, "Railway network timetabling and dynamic traffic management," *International Journal of Civil Engineering*, vol. 8, no. 1, pp. 19–32, 2010.

[21] M. Samà, C. Meloni, A. D'Ariano, and F. Corman, "A multi-criteria decision support methodology for real-time train scheduling," *Journal of Rail Transport Planning & Management*, vol. 5, no. 3, pp. 146–162, 2015.

[22] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company, 2004.

[23] M. Samà, P. Pellegrini, A. D'Ariano, J. Rodriguez, and D. Pacciarelli, "A routing filter for the real-time railway traffic management problem based on ant colony optimization," *Transportation Research Procedia*, vol. 10, pp. 534–543, 2015.

[24] M. Guntsch and M. Middendorf, "Pheromone modification strategies for ant algorithms applied to dynamic TSP," in *Applications of Evolutionary Computing*, pp. 213–222, Springer, 2001.

[25] C. Eyckelhof and M. Snoek, "Ant systems for a dynamic tsp," in *International Workshop on Ant Algorithms*, pp. 88–99, Springer, 2002.

[26] M. Mavrovouniotis and S. Yang, "A memetic ant colony optimization algorithm for the dynamic travelling salesman problem," *Soft Computing*, vol. 15, no. 7, pp. 1405–1425, 2011.

[27] M. Mavrovouniotis and S. Yang, "Ant colony optimization algorithms with immigrants schemes for the dynamic travelling salesman problem," in *Evolutionary Computation for Dynamic Optimization Problems* (S. Yang and X. Yao, eds.), vol. 490 of *Studies in Computational Intelligence*, pp. 317–341, Springer Berlin Heidelberg, 2013.

[28] M. Mavrovouniotis, S. Yang, and X. Yao, "Multi-colony ant algorithms for the dynamic travelling salesman problem," in *Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2014 IEEE Symposium on*, pp. 9–16, IEEE, 2014.

[29] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001.

[30] Network Rail, "Data feeds," 2015. [Accessed on 1st October 2015].

[31] F. Corman, A. D'Ariano, A. Marra, D. Pacciarelli, and M. Samà, "Integrating train scheduling and delay management in real-time railway traffic control," *Transportation Research Part E: Logistics and Transportation Review*, 2016.

[32] A. D'Ariano, D. Pacciarelli, and M. Pranzo, "A branch and bound algorithm for scheduling trains in a railway network," *European Journal of Operational Research*, vol. 183, no. 2, pp. 643–657, 2007.

[33] M. Samà, A. D'Ariano, D. Pacciarelli, and F. Corman, "Lower and upper bound algorithms for the real-time train scheduling and routing problem in a railway network," *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 215–220, 2016.

[34] J. Pachl, *Railway Operation and Control*. VTD Rail Publishing, Mountlake Terrace, USA, second ed., 2009.

[35] A. Mascis, D. Pacciarelli, and M. Pranzo, "Scheduling models for short-term railway traffic optimisation," in *Computer-aided systems in public transport*, pp. 71–90, Springer, 2008.

[36] M. Guntsch and M. Middendorf, "Applying population based ACO to dynamic optimization problems," in *Ant Algorithms*, pp. 111–122, Springer, 2002.

[37] T. Stützle and H. Hoos, "MAX-MIN ant system and local search for the traveling salesman problem," in *Evolutionary Computation, 1997., IEEE International Conference on*, pp. 309–314, Apr 1997.

[38] J. Eaton, S. Yang, and M. Mavrovouniotis, "Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays," *Soft Computing*, vol. 20, pp. 2951–2966, August 2016.

[39] M. Dorigo and L. Gambardella, "Ant colonies for the travelling salesman problem," *BioSystems*, vol. 43, no. 2, pp. 73–81, 1997.

[40] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 53–66, 1997.

[41] Y. Semet and M. Schoenauer, "An efficient memetic, permutation-based evolutionary algorithm for real-world train timetabling," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, pp. 2752–2759, IEEE, 2005.

[42] F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo, "Evaluation of green wave policy in real-time railway traffic management," *Transportation Research Part C: Emerging Technologies*, vol. 17, no. 6, pp. 607–616, 2009.

[43] A. D'Ariano, *Improving real-time train dispatching: models, algorithms and applications*. No. T2008/6, Netherlands TRAIL Research School, 2008.

[44] P. Pellegrini, G. Marlière, R. Pesenti, and J. Rodriguez, "Recife-milp: An effective milp-based heuristic for the real-time railway traffic management problem," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2609–2619, 2015.

[45] C. Goodman and R. Takagi, "Dynamic re-scheduling of trains after disruption," *WIT Transactions on The Built Environment*, vol. 74, 2004.

[46] S. Mladenovic, S. Veskovic, I. Branovic, S. Jankovic, and S. Acimovic, "Heuristic based real-time train rescheduling system," *Networks*, vol. 67, no. 1, pp. 32–48, 2016.

[47] T. Ho and T. Yeung, "Railway junction conflict resolution by genetic algorithm," *Electronics Letters*, vol. 36, no. 8, pp. 771–772, 2000.

[48] T. Ho and T. Yeung, "Railway junction traffic control by heuristic methods," *IEE Proceedings-Electric Power Applications*, vol. 148, no. 1, pp. 77–84, 2001.

[49] L. Chen, F. Schmid, M. Dasigi, B. Ning, C. Roberts, and T. Tang, "Real-time train rescheduling in junction areas," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 224, no. 6, pp. 547–557, 2010.

[50] S. Dündar and İ. Şahin, "Train re-scheduling with genetic algorithms and artificial neural networks for single-track railways," *Transportation Research Part C: Emerging Technologies*, vol. 27, pp. 1–15, 2013.

[51] L. Chen, C. Roberts, F. Schmid, and E. Stewart, "Modeling and solving real-time train rescheduling problems in railway bottleneck sections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1896–1904, 2015.

[52] A. D'Ariano, M. Pranzo, and I. Hansen, "Conflict resolution and train speed coordination for solving real-time timetable perturbations," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 8, pp. 208–222, June 2007.

[53] A. D'Ariano and M. Pranzo, "An advanced real-time train dispatching system for minimizing the propagation of delays in a dispatching area under severe disturbances," *Networks and Spatial Economics*, vol. 9, no. 1, pp. 63–84, 2009.

[54] F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo, "A tabu search algorithm for rerouting trains during rail operations," *Transportation Research Part B: Methodological*, vol. 44, no. 1, pp. 175–192, 2010.

[55] F. Corman, A. D'Ariano, M. Pranzo, and I. Hansen, "Effectiveness of dynamic reordering and rerouting of trains in a complicated and densely occupied station area," *Transportation Planning and Technology*, vol. 34, no. 4, pp. 341–362, 2011.

[56] F. Corman, I. D'Ariano, A.and Hansen, and D. Pacciarelli, "Optimal multi-class rescheduling of railway traffic," *Journal of Rail Transport Planning & Management*, vol. 1, no. 1, pp. 14–24, 2011.

[57] A. D'Ariano, M. Samà, P. D'Ariano, and D. Pacciarelli, "Evaluating the applicability of advanced techniques for practical real-time train scheduling," *Transportation Research Procedia*, vol. 3, pp. 279–288, 2014.

[58] J. Rodriguez, "A constraint programming model for real-time train scheduling at junctions," *Transportation Research Part B: Methodological*, vol. 41, no. 2, pp. 231–245, 2007.

[59] J. Törnquist, "Railway traffic disturbance management-an experimental analysis of disturbance complexity, management objectives and limitations in planning horizon.," *Transportation Research Part A: Policy and Practice*, vol. 41, no. 3, pp. 249 – 266, 2007.

[60] M. Mazzarello and E. Ottaviani, "A traffic management system for real-time traffic optimisation in railways," *Transportation Research Part B: Methodological*, vol. 41, no. 2, pp. 246–274, 2007.

[61] B. Khosravi, J. Bennell, and C. Potts, "Train scheduling and rescheduling in the UK with a modified shifting bottleneck procedure," in *OASIcs-OpenAccess Series in Informatics*, vol. 25, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[62] J. Espinosa-Aranda and R. García-Ródenas, "A demand-based weighted train delay approach for rescheduling railway networks in real time," *Journal of Rail Transport Planning & Management*, vol. 3, no. 1, pp. 1–13, 2013.

[63] F. Corman, D. Pacciarelli, A. D'Ariano, and M. Samà, "Railway traffic rescheduling with minimization of passengers? discomfort," in *Proceedings of 6th International Conference on Computational Logistics (ICCL 2015)*, 2015.

[64] J. Törnquist Krasemann, "Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances," *Transportation Research Part C: Emerging Technologies*, vol. 20, no. 1, pp. 62–78, 2012.

[65] J. Törnquist and J. Persson, "N-tracked railway traffic re-scheduling during disturbances," *Transportation Research Part B: Methodological*, vol. 41, no. 3, pp. 342–362, 2007.

[66] T. Dollevoet, F. Corman, A. D'Ariano, and D. Huisman, "An iterative optimization framework for delay management and train scheduling," *Flexible Services and Manufacturing Journal*, vol. 26, no. 4, pp. 490–515, 2014.

[67] L. Meng and X. Zhou, "Simultaneous train rerouting and rescheduling on an n-track network: A model reformulation with network-based cumulative flow variables," *Transportation Research Part B: Methodological*, vol. 67, pp. 208–234, 2014.

[68] J. Törnquist and J. Persson, "Train traffic deviation handling using tabu search and simulated annealing," in *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, pp. 73a–73a, Jan 2005.

[69] D. Šemrov, R. Marsetič, M. Žura, L. Todorovski, and A. Srdic, "Reinforcement learning approach for train rescheduling on a single-track railway," *Transportation Research Part B: Methodological*, vol. 86, pp. 250–267, 2016.

[70] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *European Journal of Operational Research*, vol. 143, no. 3, pp. 498–517, 2002.

[71] T. Ho, J. Norton, and C. Goodman, "Optimal traffic control at railway junctions," *IEE Proceedings-Electric Power Applications*, vol. 144, no. 2, pp. 140–148, 1997.

[72] M. Dessouky and S. Mi, *Dynamic Scheduling of Trains in Densely Populated Congested Areas*. Daniel J. Epstein Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, California, 2011.

[73] L. Dai, G. Nicholson, D. Kirkwood, and C. Roberts, "Dynamic train rescheduling using alternating algorithms," in *2016 IEEE International Conference on Intelligent Rail Transportation (ICIRT)*, pp. 1–7, Aug 2016.

[74] J. Törnquist Krasemann, "Computational decision-support for railway traffic management and associated configuration challenges: An experimental study," *Journal of Rail Transport Planning & Management*, vol. 5, no. 3, pp. 95–109, 2015.

[75] T. Albrecht and S. Oettich, "A new integrated approach to dynamic schedule synchronization and energy-saving train control," *WIT Transactions on The Built Environment*, vol. 61, 2002.

[76] C. Walker, J. Snowdon, and D. Ryan, "Simultaneous disruption recovery of a train timetable and crew roster in real time," *Computers & Operations Research*, vol. 32, no. 8, pp. 2077–2094, 2005.

[77] M. Flamini and D. Pacciarelli, "Real time management of a metro rail terminus," *European Journal of Operational Research*, vol. 189, no. 3, pp. 746–761, 2008.

[78] M. Schachtebeck and A. Schöbel, "IP-based techniques for delay management with priority decisions.," *ATMOS*, vol. 8002, 2008.

[79] T. Dollevoet, D. Huisman, L. Kroon, M. Schmidt, and A. Schöbel, "Delay management including capacities of stations," *Transportation Science*, vol. 49, no. 2, pp. 185–203, 2015.

[80] R. Acuna-Agost, P. Michelon, D. Feillet, and S. Gueye, "Sapi: Statistical analysis of propagation of incidents. a new approach for rescheduling trains after disruptions," *European Journal of Operational Research*, vol. 215, no. 1, pp. 227–243, 2011.

[81] G. Caimi, M. Fuchsberger, M. Laumanns, and M. Lüthi, "A model predictive control approach for discrete-time rescheduling in complex central railway station areas," *Computers & Operations Research*, vol. 39, no. 11, pp. 2578–2593, 2012.

[82] L. Veelenturf, M. Kidd, V. Cacchiani, L. Kroon, and P. Toth, "A railway timetable rescheduling approach for handling large-scale disruptions," *Transportation Science*, 2015.

[83] A. Toletti, V. De Martinis, and U. Weidmann, "What about train length and energy efficiency of freight trains in rescheduling models?," *Transportation Research Procedia*, vol. 10, pp. 584–594, 2015.

[84] M. Tamannaei, M. Saffarzadeh, A. Jamili, and S. Seyedabrishami, "A novel train rescheduling approach in double-track railways: Optimization model and solution method based on simulated annealing algorithm," *International Journal of Civil Engineering*, vol. 14, no. 3, pp. 139–150, 2016.

[85] S. Umiliacchi, G. Nicholson, N. Zhao, F. Schmid, and C. Roberts, "Delay management and energy consumption minimisation on a single-track railway," *IET Intelligent Transport Systems*, vol. 10, no. 1, pp. 50–57, 2016.

[86] J. Yin, T. Tang, L. Yang, Z. Gao, and B. Ran, "Energy-efficient metro train rescheduling with uncertain time-variant passenger demands: An approximate dynamic programming approach," *Transportation Research Part B: Methodological*, vol. 91, pp. 178–210, 2016.

[87] E. Zitzler, D. Brockhoff, and L. Thiele, "The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration," in *Evolutionary multi-criterion optimization*, pp. 862–876, Springer, 2007.

[88] C. García-Martínez, O. Cordón, and F. Herrera, "A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp," *European Journal of Operational Research*, vol. 180, no. 1, pp. 116–148, 2007.

[89] D. Angus, "Crowding population-based ant colony optimisation for the multi-objective travelling salesman problem," in *Computational Intelligence in Multicriteria Decision Making, IEEE Symposium on*, pp. 333–340, IEEE, 2007.

[90] L. Bui, J. Whitacre, and H. Abbass, "Performance analysis of elitism in multi-objective ant colony optimization algorithms," in *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pp. 1633–1640, June 2008.

[91] M. López-Ibáñez and T. Stützle, "The automatic design of multi-objective ant colony optimization algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 16, pp. 861–875, Dec 2012.

[92] K. Liangjun, Z. Qingfu, and R. Battiti, "MOEA/D-ACO: A multiobjective evolutionary algorithm using decomposition and ant colony," *Cybernetics, IEEE Transactions on*, vol. 43, pp. 1845–1859, Dec 2013.

[93] I. Alaya, C. Solnon, and K. Ghedira, "Ant colony optimization for multi-objective optimization problems," in *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, vol. 1, pp. 450–457, Oct 2007.

[94] M. Guntsch and M. Middendorf, "Solving multi-criteria optimization problems with population-based aco," in *Evolutionary Multi-Criterion Optimization* (C. Fonseca, P. Fleming, E. Zitzler, L. Thiele, and K. Deb, eds.), vol. 2632 of *Lecture Notes in Computer Science*, pp. 464–478, Springer Berlin Heidelberg, 2003.

[95] S. Iredi, D. Merkle, and M. Middendorf, "Bi-criterion optimization with multi colony ant algorithms," in *Evolutionary Multi-Criterion Optimization*, pp. 359–372, Springer, 2001.

[96] H. Panahi, M. Rabbani, and R. Tavakkoli-Moghaddam, "Solving an open shop scheduling problem by a novel hybrid multi-objective ant colony optimization," in *Hybrid Intelligent Systems, 2008. HIS '08. Eighth International Conference on*, pp. 320–325, Sept 2008.

[97] S. Chaharsooghi and A. Kermani, "An effective ant colony optimization algorithm (aco) for multi-objective resource allocation problem (morap)," *Applied Mathematics and Computation*, vol. 200, no. 1, pp. 167–177, 2008.

[98] A. Berrichi, F. Yalaoui, L. Amodeo, and M. Mezghiche, "Bi-objective ant colony optimization approach to optimize production and maintenance scheduling," *Computers & Operations Research*, vol. 37, no. 9, pp. 1584–1596, 2010.

[99] B. Barán and M. Schaerer, "A multiobjective ant colony system for vehicle routing problem with time windows.," in *Applied Informatics*, pp. 97–102, 2003.

[100] C. Mariano and E. Morales, "A multiple objective ant-q algorithm for the design of water distribution irrigation networks," *Instituto Mexicano de Tecnología del Agua, Technical Report HC-9904*, 1999.

[101] K. Doerner, W. Gutjahr, R. Hartl, C. Strauss, and C. Stummer, "Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection," *Annals of operations research*, vol. 131, no. 1-4, pp. 79–99, 2004.

[102] D. Pinto and B. Barán, "Solving multiobjective multicast routing problem with a new ant colony optimization approach," in *Proceedings of the 3rd international IFIP/ACM Latin American conference on Networking*, pp. 11–19, ACM, 2005.

[103] S. Alam, L. Bui, H. Abbass, and M. Barlow, "Pareto meta-heuristics for generating safe flight trajectories under weather hazards," in *Simulated Evolution and Learning*, vol. 4247 of *Lecture Notes in Computer Science*, pp. 829–836, Springer Berlin Heidelberg, 2006.

[104] Y. Yang, G. Wu, J. Chen, and W. Dai, "Multi-objective optimization based on ant colony optimization in grid over optical burst switching networks," *Expert Systems with Applications*, vol. 37, no. 2, pp. 1769–1775, 2010.

[105] M. Chica, Ó. Cordón, S. Damas, and J. Bautista, "Multiobjective constructive heuristics for the 1/3 variant of the time and space assembly line balancing problem: ACO and random greedy search," *Information Sciences*, vol. 180, no. 18, pp. 3465–3487, 2010.

[106] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1230–1242, 2013.

[107] C. Colson, M. Nehrir, and S. Pourmousavi, "Towards real-time microgrid power management using computational intelligence methods," in *Power and Energy Society General Meeting, 2010 IEEE*, pp. 1–8, IEEE, 2010.

[108] D. Kirkwood, C. Roberts, and F. Schmid, "Validation of railway network simulation software," in *Proceedings of the 5th International Conference on Railway Operations Modelling and Analysis*, (Copenhagen), IAROR, 2013.

[109] R. Watson, *Train planning in a fragmented railway - a British perspective.* PhD thesis, Loughborough University, 2008.

[110] National Rail Enquiries, "Stations services and facilities," January 2015. [Accessed on 15th October 2015].

[111] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes in dynamic environments," in *Parallel Problem Solving from Nature, PPSN XI*, pp. 371–380, Springer, 2010.

[112] S. Van Der Zwaan and C. Marques, "Ant colony optimisation for job shop scheduling," in *Proceedings of the Third Workshop on Genetic Algorithms and Artificial Life (GAAL 99)*, 1999.

[113] J. Grefenstette, "Genetic algorithms for changing environments," in *PPSN*, vol. 2, pp. 137–144, 1992.

[114] S. Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, pp. 1115–1122, ACM, 2005.

[115] S. Yang, "Genetic algorithms with elitism-based immigrants for changing optimization problems," in *Workshops on Applications of Evolutionary Computation*, pp. 627–636, Springer, 2007.

[116] S. Yang, "Genetic algorithms with memory-and elitism-based immigrants in dynamic environments," *Evolutionary Computation*, vol. 16, no. 3, pp. 385–416, 2008.

[117] M. Mavrovouniotis and S. Yang, "Memory-based immigrants for ant colony optimization in changing environments," in *European Conference on the Applications of Evolutionary Computation*, pp. 324–333, Springer, 2011.

[118] L. Gambardella and M. Dorigo, "An ant colony system hybridized with a new local search for the sequential ordering problem," *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 237–255, 2000.

[119] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, pp. 182–197, Apr 2002.

[120] G. Nicholson, D. Kirkwood, C. Roberts, and F. Schmid, "Benchmarking and evaluation of railway operations performance," *Journal of Rail Transport Planning & Management*, vol. 5, no. 4, pp. 274–293, 2015.

[121] J. Eaton, S. Yang, and M. Gongora, "Ant colony optimisation for simulated dynamic multi-objective railway junction rescheduling.," *IEEE Transactions on Intelligent Transportation Systems (in press)*, 2017.

[122] C. Fonseca and P. Fleming, "On the performance assessment and comparison of stochastic multiobjective optimizers," in *Parallel Problem Solving from Nature — PPSN IV* (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), vol. 1141 of *Lecture Notes in Computer Science*, pp. 584–593, Springer Berlin Heidelberg, 1996.

[123] J. Eaton and S. Yang, "Railway platform reallocation after dynamic perturbations using ant colony optimisation," in *In IEEE Symposium Series on Computational Intelligence, 2016 Proceedings of the*, IEEE, 2016.

[124] C. Hopper, "Response to draft consultation document East Midlands route study," 2016. Accessed on 20th Jan 2016.

[125] Open Rail Data wiki, "Cif codes," 2016. [Online; accessed 23-March-2016].

[126] M. Bridge, ed., *Track Atlas of Mainland Britain.* Platform 5 Publishing Ltd, Sheffield, England, 2nd ed., 2012.

[127] N. Rail, "Timetable planning rules, east midlands," 2018. [Accessed on 12th December 2016].

[128] M. Mavrovouniotis and S. Yang, "Empirical study on the effect of population size on max-min ant system in dynamic environments," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pp. 853–860, IEEE, 2016.

[129] M. Guntsch, M. Middendorf, and H. Schmeck, "An ant colony optimization approach to dynamic TSP," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pp. 860–867, Morgan Kaufmann Publishers Inc., 2001.

[130] W. Xiang and H. Lee, "Ant colony intelligence in multi-agent dynamic manufacturing scheduling," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 1, pp. 73–85, 2008.

[131] P. Renna, "Job shop scheduling by pheromone approach in a dynamic environment," *International Journal of Computer Integrated Manufacturing*, vol. 23, no. 5, pp. 412–424, 2010.

[132] R. Zhou, A. Nee, and H. Lee, "Performance of an ant colony optimisation algorithm in dynamic job shop scheduling problems," *International Journal of Production Research*, vol. 47, no. 11, pp. 2903–2920, 2009.

[133] M.-S. Lu and R. Romanowski, "Multi-contextual ant colony optimization of intermediate dynamic job shop problems," *The International Journal of Advanced Manufacturing Technology*, vol. 60, no. 5-8, pp. 667–681, 2012.

[134] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

Appendices

Appendix A

Comparison of Best-So-Far Ant Replacement Schemes

In this work, the best-so-far ant is the the best solution found since the beginning of the current dynamic change. Three best-so-far ant replacement schemes are investigated; RS1, RS2 and RS3. The details of each are given below.

- RS1: ant^{bs} is always replaced by ant^{bi} if ant^{bi}'s objective value (total delay) is less than ant^{bs}'s objective value
- RS2: ant^{bs} is only replaced with ant^{bi} if both the objective value (total delay) and the platform displacement for ant^{bi} is less than that for ant^{bs}
- RS3: ant^{bs} is always replaced with ant^{bi} if the objective value (total delay) for ant^{bi} is less than that for ant^{bs}. If the objective values for both ants are equal the amount of platform displacement is taken into account. If the platform displacement for ant^{bi} is less than that for ant^{bs}, ant^{bs} is replaced with ant^{bi}.

Table A.1 summarises the combinations of the heuristic and replacement schemes investigated. Figs. A.1 and A.2 show box plots of the average delay and platform displacement respectively for delay scenario $m = 20$, $f = 20$, for each replacement scheme. In each case, the delay and platform displacement are averaged over all dynamic changes. The horizontal line within the box represents the median, the top and bottom whiskers represent the maximum and the minimum values respectively, while the top and bottom of the box represent the third quartile and first quartile respectively.

With regards to platform displacement, RS2-H has the most positive effect on reducing platform displacement while RS1, which does not guide the algorithm in anyway towards reducing the number of platform changes, performs worse. RS1-H,

Table A.1: Details of the algorithms under investigation

Name	Heuristic Employed	Replacement Scheme
RS1-H	Yes	RS1
RS2-H	Yes	RS2
RS3-H	Yes	RS3
RS1	No	RS1

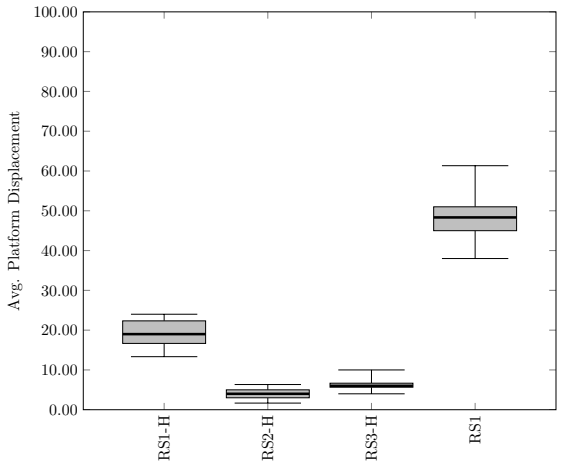


Figure A.1: Comparison of best-so-far ant replacement schemes for $m = 20$, $f = 20$

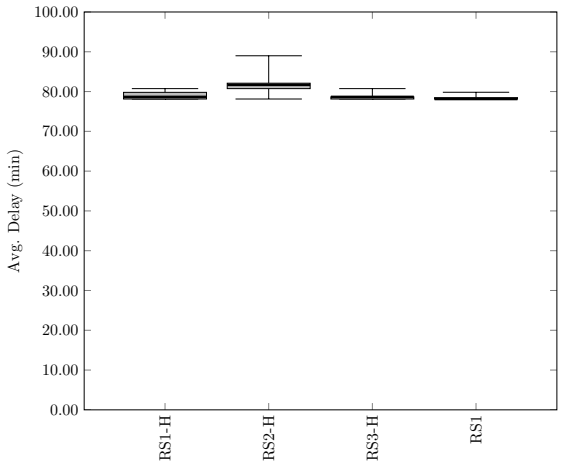


Figure A.2: Comparison of best-so-far ant replacement schemes for $m = 20$, $f = 20$

which uses the platform displacement heuristic but uses the original RS1 replacement scheme, also performs poorly on platform displacement compared to the algorithms that make use of either RS2 or RS3 replacment schemes. From Fig. A.2, it is apparent that although RS2-H gives the best values in terms of platform displacement its effect may be too strong and its overemphasis on reducing the number of platform

changes appears to restrict it in finding good results in terms of delay. For this reason, algorithm RS3-H is chosen to be implemented for the following experiments, as it provides a balance between reducing unnecessary platform changes and reducing delay.

Appendix B

First Free Platform Heuristic (FFP)

Algorithm 9 FFP

```
1: Input G                                          ▷ A list of platform gaps
2: Input n                                          ▷ number of platform gaps
3: Input T_d                                        ▷ The delayed train
4: PopulateG                                        ▷ Identify the gaps and store in G
5: Sort G in ascending platform displacement cost
6: for (i=0 to i=n-1) do
7:     if (T_d.delayedArrivalTime >= G_i.startTime) and T_d.dwellTime <= G_i.gapSize) then
8:         if (T_d fits into gap) then
9:             T_d.newPlatform = G_i.platform
10:            exit loop
11:        end if
12:    else if (T_d.delayedArrivalTime < G_i.startTime ) and T_d.dwellTime <= G_i.gapSize) then
13:        delay T_d to match G_i.startTime
14:        if (T_d with additional delay fits into gap) then
15:            propagateDelay                       ▷ through all the TIPLOCs on T_d's route
16:            T_d.newPlatform = G_i.platform
17:            exit loop
18:        end if
19:    end if
20: end for
```

The first stage of the heuristic is to identify all possible gaps on the platform for the delayed train. These are gaps where the start time of the gap corresponds to the new arrival time of the delayed train and where the duration of the gap is large enough to accommodate the train's occupation of the platform track section. The

Table B.1: Results of running FFP with a delay frequency of 10 min

Scenario	Change 1	Change 2	Change 3	Change 4	Change 5	Change 6
$m10f10$	yes	**no**	yes	yes	yes	yes
$m20f10$	yes	yes	**no**	yes	yes	yes
$m30f10$	yes	yes	**no**	yes	yes	yes

constraints in Eqs. (B.1) and (B.2) need to be satisfied for a gap to be deemed a feasible gap for delayed train.

$$G_g^{start} = x_{k,b}^{arrive} \quad k \in E_r, b \in B_r \tag{B.1}$$

$$G_g^{duration} >= x_{k,b}^{arrive} - x_{k,b}^{depart} \quad k \in E_r, b \in B_r \tag{B.2}$$

where G is the set of all suitable gaps for the delayed train and g is the index of a gap ($g \in G$), G_g^{start} and $G_g^{duration}$ are the start time and duration of gap G_g respectively. A gap is also deemed suitable for a train if G_g^{start} is later than the train's delayed arrival time but $G_g^{duration}$ is large enough to cover the train's track occupation plus the extra delay needed to be added to the train to force it to arrive at G_g^{start}. In this case the train is delayed further, to allow it to fit into the gap, and the extra delay is propagated along all of the timing points on the train's route. Once all suitable gaps are identified, FFP selects the gap on the platform that is closest to the train's original platform. The implementation of this heuristic is detailed in Algorithm 9.

This comparison also takes into account Network Rail's Timetable Planning Rules (see Section 6.2). A gap is only deemed suitable if placing the train in that gap means it would still obey the platform reoccupation rules. That is there must be at least three min between trains travelling in the same direction and four min between trains travelling in opposite directions. However, if both trains are HST trains travelling in opposite directions there must be five min between platform reoccupation.

Tables B.1, B.2 and B.3 show that FFP performed poorly in the high and medium frequency delay scenarios, as it often failed to find a free platform within the timetable planning rules to place the train on. A 'no' in the table shows that there was no platform space available to place the train on in that change period.

Table B.2: Results of running FFP with a delay frequency of 20 min

Scenario	Change 1	Change 2	Change 3
$m10f20$	yes	yes	yes
$m20f20$	yes	**no**	yes
$m30f20$	yes	**no**	yes

Table B.3: Results of running FFP with a delay frequency of 30 min

Scenario	Change 1	Change 2
$m10f30$	yes	yes
$m20f30$	yes	yes
$m30f30$	yes	yes

Appendix C

Investigation into Notification and Planning Intervals

To decide on a notification period and planning horizon to use in the delay scenario experiments, different combinations of notification periods and planning horizons were investigated. In this case notification periods of 10, 30 and 60 minutes and planning horizons of 10, 30 and 60 minutes were considered. Fig. C.1 shows the outcome of these experiments, where each experiment refers to a different notification and planning combination, for example, n10p60 refers to a notification period of 10 min and a planning horizon of 60 min.

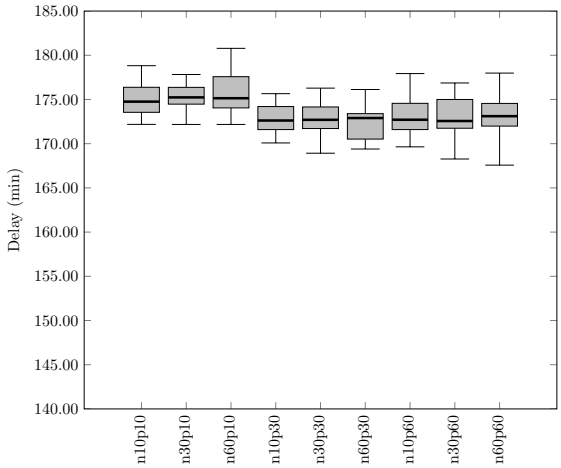


Figure C.1: Comparison of different notification and planning intervals for $m = 30$, $f = 10$

Although a Kruskal-wallis test showed no significant difference between the medians of each notification/planning combination, we can see from the box plot that longer planning horizons of 30 min and 60 min give, on average, slightly less delay than a short planning horizon of 10 min. This is to be expected as the longer the

planning horizon the more trains can be included in the problem given to the ants and the more options they have to rearrange those trains on the platforms to reduce the objective value. In contrast, the notification interval appears to have very little influence on the performance of the algorithm. For example, the first three boxes for p=10 and n=10, 30 and 60 all have similar median values. The longer the notification period, the more trains can be reshuffled before the delayed train arrives. This result suggests that rearranging trains that arrive before the delayed train has very little influence on the outcome of the algorithm and that it is trains that arrive later than the delayed train that are the important ones in terms of finding a good problem solution.