Multi-train trajectory planning

By:

Jonathan C J Goodwin

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

The University of Sheffield
Faculty of Engineering
Department of Mechanical Engineering

October 2016

Abstract

Although different parts of the rail industry may have different primary concerns, all are under increasing pressure to minimise their operational energy consumption. Advances in single-train trajectory optimisation have allowed punctuality and traction energy efficiency to be maximised for isolated trains. However, on a railway network safe separation of trains is ensured by signalling and interlocking systems, so the movement of one train will impact the movement of others. This thesis considers methodologies for multi-train trajectory planning.

First, a genetic algorithm is implemented and two bespoke genetic operators proposed to improve specific aspects of the optimisation. Compared with published results, the new optimisation is shown to increase the quality of solutions found by an average of 27.6% and increase consistency by a factor of 28. This allows detailed investigation into the effect of the relative priority given to achieving time targets or increasing energy efficiency.

Secondly, the performance of optimised control strategies is investigated in a system containing uncertainty. Solutions optimised for a system without uncertainty perform well in those conditions but their performance quickly degrades as the level of uncertainty increases. To address this, a new genetic algorithm-based optimisation procedure is introduced and shown to find robust solutions in a system with multiple different types of uncertainty. Trade-offs are explored between highly optimised trajectories that are unlikely to be achieved, and slightly less optimal trajectories that are robust to real world disturbances.

Finally, a massively parallel multi-train simulator is developed to accelerate population-based heuristic optimisations using a graphical processing unit (GPU). Execution time is minimised by implementing all parts of the simulation and optimisation on the GPU, and by designing data structure and algorithms to work efficiently together. This yields a three orders of magnitude increase in rate at which candidate control strategies can be evaluated.

Acknowledgements

Thanks to my supervisors, David Fletcher and Rob Harrison, for supporting and guiding me through the PhD process. I am also grateful to the other group members Adam, Jon, Emma and Rahi for their encouragement throughout. Meena Dasigi at Network Rail has kindly provided valuable industrial insight and thanks to Paul Richmond for introducing me to GPU computing. The E-Futures Doctoral Training Centre (and in particular Neil Lowrie) gave me the rare chance of developing my interdisciplinary studies. Most recently, I am grateful to my employers at MachineWorks for allowing me the flexibility to complete this thesis.

Lots of thanks to my parents for their endless proofreading, love and support. Thanks also to my wife Sarah for her encouragement and support, proofreading, and making sure I ate. Our church family at The Crowded House have been crucial in looking after both of us, particularly with the arrival of Thomas four months before my submission. Finally, I am grateful for Thomas' company during my late-night writing sessions.

Contents

Terminology

Adhesion - the grip between wheel and rail limiting the maximum traction force that can be applied

Block - (in signaling) the resolution to which train locations are detected and restricted in fixed block signaling

Block - (CUDA abstraction) a group of simultaneously executed threads

Closed-loop control - feedback is available on the current state of the railway network (e.g. positions and velocities of trains) which can be used to tune future control instructions

Coalesced memory access - adjacent threads in a block access adjacent memory addresses allowing full utilisation of the available memory bandwidth control point - the position at which a specific control action is defined for a train

Continuous control - instead of being controlled with notches, the traction level of many modern electric trains can be varied continuously

CUDA - an application programming interface (API) created by Nvidia which enables general purpose processing on GPUs.

Delay minutes - the total number of minute of train delay attributed to a single train

Device - a GPU

Discrete control - many older trains are controlled using discrete traction/brake notches

Fixed block - the positions of the signal blocks, which ensure a safe separation of trains, are static (i.e. fixed relative to the track)

Global memory - the GPU RAM. This is separate to the chip where execution occurs so data stored here is accessed much slower than data stored in registers or shared memory

Golden run - a perfectly controlled train journey that yields optimum energy efficiency

Grid - (CUDA abstraction) a group of independently executed blocks

Host - the computer containing the device

Kernel - (CUDA abstraction) a function that can be executed in parallel on a GPU using a grid

Line speed - the maximum speed allowed for that train of that section of railway line

Link - the track joining two stations

Moving block - the positions of the signal blocks, that ensure a safe separation of trains, are defined relative to each train (i.e. the braking distances in front of each train must not overlap)

Network - the connectivity of the railway tracks (not power distribution network)

Noise - random variation due to uncertainty

Pareto optimal - a solution where one performance metric cannot be improved without degrading another

Thread - a sequence of instructions executed in order (on a single data stream)

Training noise - random variation in some parameters during the optimisation process

Trajectory - the velocity profile of a train usually plotted against distance of time

Utilisation noise - random variation in some parameters while a pre-optimised control is being applied

Warp - (CUDA abstraction) batches of 32 threads within a block that are given exactly the same instructions during execution

List of Abbreviations

ANN - Artificial Neural Network

ATO - Automatic Train Operation

ATP - Automatic Train Protection

CP - Control Point

CPU - Central Processing Unit

DAS - Driver Advice Systems

DT - Dwell Time

ERTMS - European Rail Traffic Management System

GA - Genetic Algorithm

GPU - Graphics Processing Unit

HGA - Hierarchical Genetic Algorithm

MARK - Minimum Allele Reserve Keeper

MILP - Mixed-integer Linear Programming

MPGA - Multi-population Genetic Algorithm

NLP - Nonlinear Programming

NOC - Necessary Optimality Conditions

PMP - Pontryagin Minimum Principle

RAM - Random Access Memory

ROC - Rail Operating Centers

RNG - Random Number Generator

SIMD - Single Instruction stream Single Data stream

SISD - Single Instruction stream Multiple Data stream

List of Symbols

Chapter 2

δ	uncertainty in the application of control
γ	uncertainty in uncontrolled parameters in the system

Chapter 3

Δt	a small time step (over which constant acceleration is assumed)
D	link length, the distance between two stations
x_n	control point, given as a distance from the start of the link $(0 \leq x < D)$
y	braking point, where maximal braking must be applied to come to a stop at distance D
$\underline{x}_i$	link control strategy; made up of a list of control points. $\underline{x}_i = (x_1, x_2 ... x_n ... x_{n\_max})$ where n_max is an odd positive integer. (Note: $x_0 = 0, x_{n\_max+1} = D$)
$\underline{\boldsymbol{X}}$	network control strategy; made up of a list of link control strategies. $\underline{\boldsymbol{X}} = (\underline{x}_1, \underline{x}_2 ... \underline{x}_i ... \underline{x}_{i\_max})$ where i_max is the total number of links traversed by all trains
$T(\underline{\boldsymbol{X}})$	total traverse time of a particular network control strategy
$E(\underline{\boldsymbol{X}})$	total energy consumption of a particular network control strategy
$G(\underline{\boldsymbol{X}})$	penalty for operational interactions between different trains caused by a particular network control strategy
α	weighting between energy and time
max	the maximum value in a list of numbers
`pop_size`	the number of candidate solutions in the GA population

P_c	probability that the crossover procedure is applied an each individual in the population
P_m	probability that the mutation procedure is applied an each individual in the population
M	initial mutation distance
Tr	minimum distance between operation transitions
P_h	probability of applying the link-wise operator $h(\underline{x})$
P_i	probability that the insertion procedure is applied an each individual in the population
P_d	probability that the deletion procedure is applied an each individual in the population
P_{ins_link}	probability of inserting a pair of control points on each link
P_{del_link}	probability of deleting a pair of control points on each link
P_{ins_pair}	probability of inserting a pair of control points between control points n and $n+1$
P_{idel_pair}	probability of deleting a pair of control points n and $n+1$
`rand`	a pseudo-random number is generated within the specified range

Chapter 4

c_e	the value of energy relative to time
N	the number of re-samplings when explicitly averaging the properties of the final population

Chapter 5

Δd	a small time step (over which constant acceleration is assumed)
$floor$	round down a number to the nearest integer
v_{max}	the maximum possible speed limit
v_{min}	the lowest speed allowed between scheduled stops
$vATP$	the maximum speed which enables future speed limits to be observed
$ceil$	round up a number to the nearest integer
log	the natural logarithm of a number

Chapter 1

Introduction

1.1 Background

The long-term increase in cost and volatility of the global energy market, coupled with concern over CO_2 emissions, means that minimising energy consumption has become increasingly desirable for all industries. This is particularly true in the transport sector, which accounted for 27% of Global [1] and 39% of UK [2] energy consumption in 2011. Although different parts of the rail industry may have different primary concerns, all are under increasing pressure to minimise their operational energy consumption. However, in general, rail is already a relatively efficient transport mode, accounting for 8.7% of passenger and 9.0% of freight traffic in the UK, while constituting only 1.9% of its transport sector energy consumption in 2011. [3] This means that it is possible to reduce overall energy consumption by modal shift to rail instead of less-efficient transport modes such as road and air. Given the projected increases in transport demand, maximising the capacity of railway networks is also increasing in importance, both economically and environmentally. Although there is no definitive definition of capacity, at an operational level it is accepted that increasing the frequency of trains, while maintaining acceptable levels of punctuality, constitutes an increase in capacity. UK rail industry has summarised the situation via the 4 C's (capacity, cost, carbon, and customers) identified in [4] and through the Rail Technical Strategy[5].

Operational methods for minimising traction energy consumption and maximising punctuality are often preferable to upgrades in network infrastructure and/or rolling stock. Physical improvements usually require large capital investment and/or only improve performance in a very specific way. In contrast, operational improvements (for example timetabling, rescheduling, train control) can be easier and less expensive to introduce and have the potential to affect several different performance measures. In Great Britain, improvements in operation have been incentivised through the setting of ambitious targets by regulatory organisations and brought to bear through a system of financial penalties and incentives. For example, if a train averages 15 kWh/km over the 250 km journey from Sheffield to London[6], then the total energy consumption will be 3750 kWh costing approximately £560 (0.15 £/kWh). However, at the time of writing, a single first class ticket for this journey may cost as much as £265[7]. Also, train operators may be charged over £100 for every minute of train delay attributed to them[8]. So, while reducing energy consumption of rail operations is desirable, in practice the incentive for doing so can be small relative to the incentives for maintaining punctual operation.

In all real systems there is some degree of uncertainty originating from both internal and external sources. This makes maintaining punctual operation more difficult. While some perturbations may be very large (e.g. infrastructure or rolling stock failure) most are small (e.g. a few seconds caused by an obstruction delaying door closure) allowing scheduled operation to be recovered. However, if services are running close to maximum capacity, with short headway distances and very little slack time built into the schedule, then even small perturbations may cause problems. Also, it is important to remember that generally trains do not operate in isolation. Instead, typically a number of trains move around a network, with safe separation of trains ensured by a signalling and interlocking systems. A direct consequence of the signalling system is that the movement of one train can effect the movement of other trains. So, as the density of operations is increased, the likelihood of perturbations propagating across the network also increases.

1.2 Scope of work

In this thesis, consideration is given to increasing the energy efficiency of railway networks through better operational control. This is a very large topic and could include a number of different research areas: trajectory planning, automatic train control, timetabling and re-scheduling. Since all of these are well developed areas in their own right, and are also yet to be fully integrated with each other, the scope of this thesis must be clearly defined. Investigations in this work will consider:

- Multiple trains (not just a single train). As will be seen in section 2.1, there is already a large body of work considering trajectory optimisation for a single train operating in isolation. While this may be sufficient to optimise some systems, McClanachan and Cole [9] observe that "If the journey time of one train is extended to save energy, then this could aversely influence the schedules and energy usage of other trains on the same network." This means to maximise the performance of a whole system, the impact of interactions between trains should be taken into account and the movement of all trains should be optimised together.

- Trajectory optimisation (not scheduling). Traction energy consumption and degree to which trains interact are determined by both their scheduled operations and the specific trajectories they follow in order to implement the schedule. This means, although they are usually addressed separately, scheduling and trajectory optimisation are really part of the same overall problem. However, to increase the tractability of this problem this thesis will explore methods that explicitly optimise the train trajectories.

- Planning (not real-time control). The actual trajectory each train follows is determined by both its theoretical target trajectory and the control used to implement this. Good control must react to the changing state of the system while seeking an overall goal, for example optimising energy consumption or punctuality. This thesis will not focus on real time optimisation/control but on techniques used in the 'off-line' planning of train trajectories.

Therefore, this thesis will consider methodologies for multi-train trajectory planning with the aim of maximising punctuality and traction energy efficiency.

1.3 Thesis structure

Chapter 2 provides a review of trajectory optimisation literature. Both single- and multi-train trajectory planning are considered, along with some robust optimisation techniques.

Chapter 3 documents the development of a new multi-train trajectory optimisation, building on a state of the art model. This provides a, genetic algorithm based, multi-train trajectory optimisation. Validation of the improved multi-train simulator is detailed before going on to investigate the performance of the optimisation. Several new algorithms are proposed to address specific limitations in the optimisation, and the effectiveness of the resulting optimisations is quantified. The improved optimisation consistency allows a more detailed investigation of the effect of varying the weighting between different objectives in the cost function.

In chapter 4 a new approach to multi-train trajectory optimisation is developed to find robust solutions in the presence of uncertainty. The effectiveness of this robust optimisation was investigated in the presence of two different types of uncertainty: the accuracy of control point application, and variation in station dwell times. First, solutions are found with robustness with respect to a single type of uncertainty; then with robustness to both types of uncertainty simultaneously. Finally, the performance of the robust solutions is compared to the estimated performance of closed-loop control.

Chapter 5 describes the development of a massively parallel multi-train simulator. This is designed to accelerate population based heuristic optimisations on a desktop computer by using a GPU. All parts of the simulation and optimisation are implemented on the GPU, removing the need to slow memory transfers between the host and device. Also, the data structure and algorithms proposed are designed together, to minimise execution time by maximising coalesced memory access. Soft constraints allow decoupling of movement simulation and headway checking. This, in combination with a distance step based approach to modelling vehicle movements, allows an efficient parallelisation strategy to be adopted. The new simulation is then validated through a sensitivity analysis of Δd, the size of the distance step used when modelling vehicle movement, and the rate of simulation investigated.

Finally, chapter 6 draws together the findings of this thesis and explores topics of further work.

Chapter 2

Literature review

Given the scope detailed in Section 1.2 this literature review focusses on multi-train trajectory planning. For a general review of train trajectory optimisation literature readers are directed to [10] and [9], the latter of which is focused on freight applications where specific issues such as in-train dynamics are of greater importance. There is also a great deal of work in the area of optimal rail timetable creation and also rescheduling, where the aim is usually to recover from perturbations back to the timetabled service. These are both closely related to the multi-train trajectory planning problem; they too must take into account interactions between trains, and ultimately their outputs (scheduled journey times) are the other major constraints when minimising journey energy use. McClanachan and Cole [9] did not find “a truly integrated scheduler and train control optimizer”, which was stated as necessary “to completely optimize a whole railway system”. Therefore, only works which explicitly consider the optimisation of train velocity profiles, with respect to energy consumption, have been considered here. Additionally, it is generally accepted that other progress may have been made in the area of train trajectory optimisation, but has not been disseminated due to its commercial value.

The trajectory optimisation problem for a single train is usually approached in two ways, either analytically or numerically. Section 2.1.1 contains a discussion of analytical results, for which specific optimal train trajectories are usually found using Nonlinear Programming (NLP) techniques. Next, methods which use searching are discussed in Section 2.1.2, with particular detail given to genetic algorithms as these are more relevant to the method of multi-train optimisation investigated. Section 2.2 contains a discussion of works that specifically consider the multi-train problem. Section 2.3 looks at robust optimisation and its application to trajectory planning. Finally, the approach taken in this thesis is outlined in Section 2.4 and set in the context of the existing literature.

2.1 Single-train trajectory planning

2.1.1 Analytical results

Interest in optimising the trajectory of trains to minimise their energy consumption began in the late 1960s. By application of the Pontryagin Minimum Principle (PMP), Ichikawa [11] solved the control problem for a linear train model (resistance proportional to velocity) finding the optimal control strategy to have four phases: full power, constant speed, coasting, and maximum braking. This assumed a flat track and did not consider line speeds. In 1985 Asnis et al. [12] used PMP to solve the same control problem more generally, this time with an arbitrary resistance force and varying levels of regenerative braking efficiency. The results are consistent with those of Ichikawa but add that, in the case of 100% efficient regenerative braking there will be no coasting phase. Howlett [13] also verified these results in 1990, and reformulated

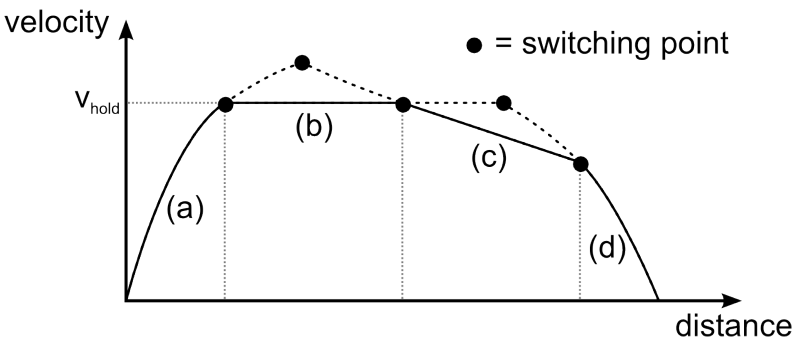


Figure 2.1: The four phases of the optimal velocity profile on flat track: (a) maximum traction, (b) cruising, (c) coasting, (d) maximum braking. Control phases (b) and/or (c) may not occur under certain conditions leading to the trajectories shown by dotted lines.

the problem in terms of finding the location of the switching points between the different phases (see Figure 2.1).

To allow useful application in most real world scenarios these idealised models must be extended to include both variable speed limits and gradients. Analytical solutions to these, more complex, problems have been formulated in two different ways - those assuming trains with continuous control (where the acceleration or traction power can take any value within the limits) and those assuming discrete control (where the traction power can take only a finite number values).

In 2000, Khmelnitsky [14] considered the continuous control problem for a system which modelled both gradients and speed limits as arbitrary variable functions of the distance. He uses PMP to show that, even with a very variable gradient profile, a complicated control sequence may contain up to four different types of mode: Maximum traction, speed holding (using traction or braking), coasting, and maximum braking. He observes that regions of speed holding tend to occur on shallow grades and steep falls, and uses properties of the optimal solution to show that the points of exit from these stable regions can be derived from the roots of a monotone function. The roots of this monotone function are then found using a dichotomous search algorithm. Liu and Golovitcher [15] obtained similar results, but state that the algebraic optimality conditions found by Golovitcher (Golovitcher 1986a,b, 1989a,b) are simpler than the differential equation that must be solved in [14]. However, Wen [16] comments that for a non-convex optimal control problem "the Necessary Optimality Conditions (NOC) like PMP do not guarantee a global optimal solution". Howlett also developed a similar approach for the problem in [17].

If a train is going up a hill sufficiently steep that it will slow down even under full power then the hill is said to be steep. Likewise, if the train speeds up when no power is applied then the slope is a steep downhill. Vu [18] showed the form of optimal control on a steep uphill is application of full power, starting before the hill and continuing after the hill until the holding speed is regained. This is illustrated in Figure 2.2. Similarly for a steep downhill Section, coasting begins before the hill and continues after the hill until the speed drops back to the holding speed.

In 1995, Howlett and Pudney [19] showed that instead of modelling a train as a distributed mass there is an equivalent control problem for a train modelled as a point mass. They also showed that discrete control can be used to approximate continuous control with an arbitrary accuracy, dependent on the number of control points. While continuous control may be a good approximation for many modern electric locomotives, there are situations which are more suited to discrete control. Diesel locomotives have different throttle notches, each giving a constant rate of fuel supply to the engine, and therefore each giving roughly constant power. This observation is commonly attributed to Benjamin et al. [20]. Much of the work on discrete optimal control has been

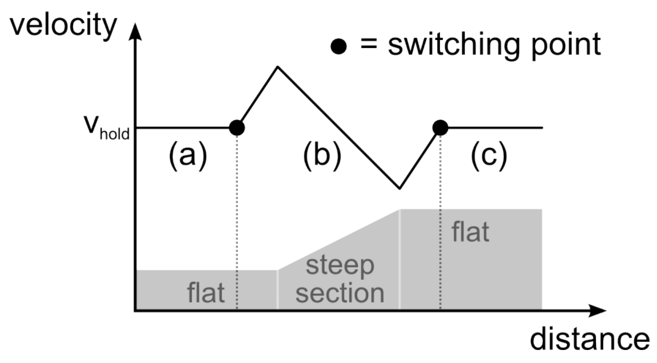


Figure 2.2: The optimal trajectory on a single steep Section follows a hold-traction-hold pattern. (a) Initially the train maintains the holding speed (v_{hold}). (b) At some point before reaching the steep Section maximum traction is applied. The train speed increases above v_{hold} before the steep slow Section, but then drops below v_{hold} on the steep Section. (c) At some point after the steep Section the speed returns to v_{hold} and the traction is reduced to hold this speed.

carried out at the University of South Australia by Howlett, Pudney, Cheng and colleagues.

2.1.2 Heuristic search methods

Due to the complexity of the modelled system, analytical methods must make approximations to simplify the problem [10], so simulation methods allow for a more realistic model. However, simulation-based optimisations are usually too slow to use where, for example, Automatic Train Operation (ATO) is desired - often light rail and metro systems. Here, other techniques have been used to find a ‘good enough’ solution in a reasonable time.

Several different types of searching algorithm have been proposed for single-train trajectory optimisation, including genetic algorithms, ant colony optimisation [21, 22, 23] and dynamic programming [24]. However, here a detailed review has only been undertaken for those based on genetic algorithms. The reasons for this are discussed in Section 2.4; after considering the multi-train trajectory optimisations and robust control literature (see Sections 2.2 and 2.3.1), it became clear genetic algorithms were most relevant to the research direction pursued in this thesis.

Genetic algorithms

Genetic algorithms (GAs) are a type of heuristic optimisation that mimic evolution by natural selection; a population of 'chromosomes' (candidate solutions) compete against each other, with information from the 'fittest' (best scoring) candidates more likely to pass to the next generation. For readers unfamiliar with GAs a brief introduction can be found in Appendix I.

In 1997, Chang's use of a GA to find the position and number of control points is widely cited as the first use of a GA to directly optimise train trajectories [25]. Each chromosome consisted of a sequence of control points for the same journey, usually alternating between traction and coasting. These were randomly initiated and used, in combination with an automatic train protection (ATP) system, to control the train during each simulation. The ATP was used to prevent line speed violations and bring the train to a stop at its destination. Each chromosome was then scored based on the modelled train energy consumption, punctuality, and jerk performance. These scores were used to select parent chromosomes in a tournament selection process, with offspring then produced using mutation, crossover, duplication, and deletion operations. If no individuals in the new population were as fit as the previous best and best alternate (different dimensioned) chromosomes then their elitist reintroduction occurred. The performance of this GA was found to compare favourably to a Fuzzy controller, described in the same paper.

In 1999, Han et al. [26] applied a GA similar to Chang's to a 920 m long metro journey and concluded that it performed better on this system than an optimisation proposed by Howlett and Pudney [19].

In 2000, Cheng et al. [27] formulated the system as an unconstrained problem, with the penalty function also including the line speed limit. He proposed optimisation by a GA followed by a local optimisation, and also detailed that his simulation model used the Runga-Kutta method.

In 2004, Colin Cole [28] optimised the trajectory of a freight train whilst considering the forces between all 107 vehicles of the train. The increased computational difficulty of this problem meant the GA had to be tailored to perform well with only a small population and few generations. Cumulative speed violations were again included in the fitness function, but this time the probability of applying different genetic operators was also adjusted as the optimisation progressed. Reproduction was performed by the breeding of the best specialists, to avoid ignoring less dominant cost function criteria early on. It was also noted that, where the fitness function contains many parameters, it is difficult to choose the weights between them without effectively negating the contribution of some parameters.

In 2004, Wong and Ho [29] applied a GA to very a simple trajectory optimisation, which found the position of up to two coasting points given a re-motoring velocity, and also stated the benefits of applying two different GA techniques. The first, Hierarchical Genetic Algorithm (HGA), allowed one gene to control the expression of other genes - in this case determining the number of coasting points. The second, Minimum-Allele-Reserve-Keeper (MARK), was intended to help fast convergence whilst allowing exploration of the search space.

In 2007, Bocharnikov et al. [30] formulated the trajectory optimisation problem as the optimisation of three parameters: K_v (re-motoring velocity, as fraction of line speed), K_f (traction force, as fraction of maximum), and

K_{br} (braking force, as fraction of maximum). As expected the GA used to optimise these parameters found $K_f = K_{br} = 1$ in the detailed example, and K_v appeared to vary depending on the fitness function weighting between energy and time.

In 2008, Landi et al. [31] used an on-board monitoring system to gather electromechanical data and build a journey specific train model, including overhead line voltage and current. This model was then used as part of a GA to optimise the duration of application of a predefined order of traction conditions: first gear (manoeuvre), series, parallel, reduced field 1, rf 2, rf 3, and coast.

In 2009, Wei et al. [32] implemented a GA with a number of differences from Chang and Sim [25]. Instead of just optimising the location of control points, for alternating traction and coasting operation, each control point also defines the control action to be used - full power, partial power, coasting, partial braking, or full braking. This allows more trajectories to be defined, some of which may be closer to the global optimum. A multi-population genetic algorithm (MPGA) is chosen to allow the algorithm to maintain multiple local solutions. Each generation a predefined number of the fittest solutions migrate to a neighbouring population. Also, within each sub-population the fitness score of each solution is negatively affected by short Hamming distances to other solutions; this is intended to maintain diversity within each sub-population. Finally, an annealing selection based genetic operator is proposed for varying the length of individual solutions.

2.2 Multi-train trajectory planning

McClanachan and Cole [9] state that, “If the journey time of one train is extended to save energy, then this could aversely influence the schedules and energy usage of other trains on the same network." Somewhat surprisingly then, there has been comparatively little work on the problem of multi-train trajectory optimisation, compared to the single-train problem. This is probably due the greatly increased complexity of the problem - a non-linear problem with many discontinuities. Three different types of interaction are considered below.

2.2.1 Headway constraints on a single line

In 2008, Acikbas and Soylemez [33] used a novel approach to multi-train optimisation. Similar to Bocharnikov et al. [30], coasting and re-motoring velocities were used as the control variables. The time and energy performance of the network could then be simulated using the SimuX software, capable of modelling a multi-train system with overhead line voltages. However, the proposed GA optimisation would have been too slow using this method of evaluation, so SimuX was used to train an Artificial Neural Network (ANN) of the system. Once trained the ANN could evaluate solutions ~900 times faster than SimuX, with an error of less than 3%, allowing optimisation by GA. Results showed optimisation of coasting points in the multi-train case actually saved less energy, for same time increase, than the single train case. This was mainly attributed to better use of regenerative braking in a system with multiple trains.

In 2009, Ding et al. [34] investigated trajectory optimisation for trains following one another under moving block signalling. The control problem was formed as minimising energy consumption, given a fixed traverse time and velocity constraints. An algorithm was stated for determining the traction/brake notches of the lead train. The following train then attempts to maintain the "scheduled time interval standard" using another predefined algorithm. No attempt was made to show the optimality of either procedure.

In 2011, Gu et al. [35] also investigated trajectory optimisation for following trains under moving-block signalling. Flat track with a constant speed limit was assumed, which simplified the problem. This meant that for normal operations, the optimal traction-cruise-coast-brake profile could be used (see Figure 2.1) and conventional non-linear programming methods used to find the switching points between phases. Live calculation of the following train's trajectory also means that the optimisation can be used for ATO, where the optimisation can handle perturbations such as the lead train stopping.

In 2011, Lu and Feng [36] optimised the trajectories of two trains under 4-aspect signalling. Instead of using ATP to ensure static velocity constraints were obeyed, soft constraints were used. The cost function incorporated a linear weighted sum of energy consumption, trip time error, static overspeed (exceeding line speed constraints) and dynamic overspeed (exceeding speed constraints ensuring the safe separation of trains). A GA was used to optimise a series of control points, which encoded both a control notch (traction, coast, brake) and the position at which this control should be applied. A two-point parallel crossover operator was adopted allowing exchange between the candidate control strategies of the lead train and following train.

In 2013, Wang et al. [37] solved the two-train problem, where one train is following another, under moving-block signalling using mixed-integer linear programming (MILP). They then extended this work in [38] to include fixed-block signalling and solution using pseudo-spectral methods. Pseudo-spectral methods were found to give slightly better results than MILP but took two orders of magnitude longer to calculate. The optimisation was also carried out using the greedy (lead train trajectory optimised independently of the second train) and the simultaneous approach. As expected, the simultaneous approach gave slightly better results but took longer to calculate. However, since the number of constraints scaled linearly with the number of train trajectories being optimised it was noted that “the computation time of the bigger [multi-train] problem will be much longer”.

In 2014, Zhao et al. [39] developed a multi-train simulator that used ATP to ensure trains obeyed both line speeds and signalling constraints. Trajectories could be controlled using train target speeds. These were then optimised using an enhanced Brute Force algorithm. Before performing the Brute Force optimisation, the range allowed for each journey’s target speed was narrowed using the target traverse times and minimum headways. A case study was used to investigate the effect of 3 different driving styles (flat-out, optimal, and cautious) under 6 different signalling systems. It was found that, for a similar energy consumption, punctuality was increased by using more advanced signalling systems. This is a good reminder that trajectories resulting from optimisations that used ATP to enforce headway constraints are heavily influenced by both the driving style and signalling system used. This study was extended in [40] to consider optimisation using an ant-colony optimisation and a genetic algorithm. Both heuristic optimisations were shown to find near optimal results and greatly reduce the computation time.

2.2.2 Electrical interactions between trains

In 2004, Albrecht [41] used a two level optimisation to minimise the total energy consumption and power peaks of a network. At a high level, synchronisation of train movements was optimised, using a GA, to give better utilisation of regenerative braking energy; at a low level, individual train trajectories were optimised using dynamic programming and a linear resistance train model. Reduced energy consumption and power peaks were reported, in a case study, when applying the higher level GA.

In 2007, Miyatake and Ko [42] stated that the model in [41] could be improved because it assumed a traction-coasting-braking trajectory and did not consider the exchange of energy between the trains. Miyatake formulated the control problem to overcome these shortcomings and developed an approximate method for solving the problem faster. This showed improved energy savings, and the interesting result that energy from braking could be used to increase the kinetic energy of a nearby train, which essentially acts as an energy storage device. Miyatake, has since taken this model further [43] comparing different optimisation techniques and considering energy storage devices.

2.2.3 Other network interactions

Although much of the scheduling literature considers other network interaction (such as stations and junctions) few trajectory optimisations could be found that considered anything other than a single line, linear system.

In 2012, Yang et al. [44] described a GA based optimisation for the multi-train trajectory problem on a branched network. The GA was similar to that proposed by Chang and Sim [25], but adapted to work on a network. The algorithms used for simulation were also given in detail, including interactions in the form of headway constraints. In the context of a branched network these headway constraints could have other effects such as delaying departure times from stations. Yang also investigated the effect that the relative importance of traverse time and energy consumption had on the solutions found.

2.3 Optimisation in uncertain systems

Most trajectory planning work (for both single and multiple trains) is focussed on finding the optimal solution for a predefined schedule. Usually control seeks to maintain the scheduled traverse time for a length of track, whilst minimising the energy consumption [9]. The problem is constrained by the dynamic performance limits of trains and restrictions on velocity and headways (imposed to ensure safe operation). The traditional formulation of the multi-train trajectory planning problem can be formalised as:

$$f(\underline{\boldsymbol{X}}) \rightarrow min \tag{2.1}$$

where $\underline{\boldsymbol{X}}$ is a control strategy for all trains on the network, and $f(\underline{\boldsymbol{X}})$ is a cost function usually based on the traverse times and energy consumptions of all train journeys.

It is assumed that if the 'optimal' control strategy is identified, then it can be implemented resulting in optimal performance of the system. However, the optimum is likely to lie on the limit of the feasibility boundary so will be very sensitive to noise [45]. In this context 'noise' refers to the many small uncertainties that most current models do not consider but which do exist in reality. These mean that if most optimised control strategies (for a noiseless system) were applied to real operation then it is likely they would not perform as well as expected, and may in-fact result in severely sub-optimal outcomes. Most of these uncertainties fall into two different groups classified by Chen et al. [46] as:

Type 1 - variations in performance caused by variations in uncontrolled parameters.

Type 2 - variations in performance caused by variations in the design (control) variables.

When controlling a train it is just not possible for all drivers to consistently follow a control strategy perfectly accurately. Automatic train control systems can achieve a very repeatable application of control, but modern systems deliberately make small variations in control to avoid problems with localised track wear in the positions where control actions are applied. Either way, there will be variation in the position of control point application equivalent to Type 2 uncertainty. Including this uncertainty the problem becomes:

$$f(\underline{\boldsymbol{X}} + \delta) \rightarrow min \tag{2.2}$$

where δ is the uncertainty in the in control application.

Likewise, in the context of train control, Type 1 uncertainties may be caused by variation in the properties of the vehicles, track being traversed or movement authority given. For example, even when composed of the same types of rolling stock, the mass, resistance, traction and braking characteristics of nominally identical trains will vary slightly. Also, some characteristics of the line, for example, the level of adhesion or overhead line voltage, are unlikely to be constant and will be affected by external factors. The combination of these uncertainties in train and track means that the expected rate of acceleration or deceleration of a train will not be known precisely in advance. Other external factors can also influence the definition of the problem being solved. Variation in station dwell times may extend or reduce the target traverse time for a journey, whilst equipment failures may cause large delays and may even stop services altogether. Including both Type 1 and 2 variations the problem becomes:

$$f(\underline{\boldsymbol{X}} + \delta, \gamma) \rightarrow min \tag{2.3}$$

where γ represents the changing operating conditions of the system.

The function $f(\underline{\boldsymbol{X}} + \delta, \gamma)$ has an effective cost function, $F(\underline{\boldsymbol{X}})$, which gives the distribution of scores for a given $\underline{\boldsymbol{X}}$ found by integration over δ and γ. Since the optimum of $f(\underline{\boldsymbol{X}})$ is not necessarily the same as the robust optimum of $F(\underline{\boldsymbol{X}})$, there is often a trade-off between the quality and the robustness of solutions. [47]

2.3.1 Techniques used in 'noisy' genetic algorithms

It has been widely noted that GAs can still effectively optimise systems in the presence of noise. Arnold [48] found that they performed particularly well on problems with both high dimensionality and high noise, when compared to a number of other direct optimisation strategies. This ability to find robust solutions in the presence of uncertainty has been demonstrated in many papers (see [47] for a comprehensive review), which consider both Type 1 and 2 uncertainties. In many practical situations the analytical form of the effective cost function, $F(\underline{\boldsymbol{X}})$, is not known but numerical approximations can be used to estimate its statistical properties. Each time the cost function in Equation (2.3) is evaluated its value will vary stochastically, but the expected (mean) score $\hat{F}(\underline{\boldsymbol{X}})$ can be estimated by averaging a number of random samples.

$$\hat{F}(\underline{\boldsymbol{X}}) = \frac{1}{N} \sum_{i=1}^{N} f(\underline{\boldsymbol{X}} + \delta, \gamma) \tag{2.4}$$

where N is the number of times the control strategy is re-sampled.

This is referred to as explicit re-sampling as each candidate solution is evaluated multiple times. The larger N used, the more likely that $\hat{F}(\underline{\boldsymbol{X}})$ is close to the actual mean but since evaluation of candidate solutions is usually the most computationally expensive part of an optimisation there is a balance between getting a good enough estimate of the expected score against the effort spent achieving this. However, GAs do not just operate on a single candidate solution, but simultaneously on a population of candidate solutions. Clustering of candidate solutions tends to occur in promising areas of the search space, making it likely that a number of similar candidate solutions are evaluated during each iteration of the algorithm. This is known as implicit re-sampling and reduces the reliance of the search on any single (potentially misleading) evaluation. In the extreme case (of infinite population) Tsutsui [49] showed that noise does not affect GAs that use a fitness proportional selection scheme. So, the effect of noise may be reduced either by explicit re-sampling (more evaluations of each candidate solution) or implicit re-sampling (increasing the population size).

2.3.2 Robust train trajectory planning

Energy efficient train operation has been studied since the late 1960s, the main results from which have been discussed in Sections 2.1 and 2.2. However, there has been almost no consideration of the effect Type 1 and 2 uncertainties may have on the optimisation. In 2013 Li et al.[50, 51] did consider the case of stochastic resistance coefficients for a single-train. They showed that on flat track the traditional coasting mode can be replaced with a 'quasicoasting' mode, where a small amount of traction or braking may be applied to offset differences in train resistance.

2.4 Evaluation and research direction

To make the numerical problems tractable it is necessary to make simplifications. Wang et al. [10] note that introducing non-linear terms into the model equations or constraints often causes difficulties for analytical solutions. For example, many of the analytical approaches to the single-train problem rely on having a constant maximum traction force. In reality, both the maximum traction and braking force are likely to vary with velocity - being limited by adhesion, power or passenger comfort. Likewise, traction efficiency is likely to vary with velocity. Whilst it is true that many of the heuristic search optimisations do not include these non-linearities, in most cases it would be trivial to introduce them. This means that, at present, real world systems can be more accurately modelled using numerical simulations than by the models underpinning analytical optimisations. Since the aim is to find solutions that are likely to be usable in real systems, numerical simulations (and consequently searching methods) are likely to give the best results. Also, few analytical results could be found for the multi-train problem.

Most of the multi-train trajectory optimisations found (see Section 2.2) either used GAs or MILP. MILP problems have the advantage that many commercial and free solvers are available, so simple problems (e.g. two trains) can be solved efficiently[37]. However, Wang et al. [37] also state that "when the number of trains taken into account increases, the size of the MILP problem will grow quickly". Even for a single train, Optimisation of the integer variables in a MILP problem is a combinatorial search so cannot be solved in polynomial time. This will limit the size of problem that can be optimised using MILP. Since searching techniques generally use simulation as a 'black box' to evaluate each control sequence, they are not as affected by the number of constraints (though to some extent this depends on how the simulation is implemented). Again, this suggests that searching techniques using numerical simulation are more likely to be easily applied to realistic problems.

Many different types of searching optimisation have been applied to train trajectory optimisation. When considering which research direction to pursue it was necessary to choose which optimisation techniques to focus on. Section 2.3.2 highlights the limited consideration of robustness when planning train trajectories. Realistic systems will always contain sources of uncertainty so it would be desirable to choose an optimisation technique that can also find robust solutions. As discussed in Section 2.3.1 GAs have shown to perform well in this respect. There are also many examples of GAs being applied to trajectory optimisation but as yet none could be found which sought to find robust solutions. For these reasons GAs were chosen as the preferred type of optimisation technique. However, it should be emphasised that GAs are not necessarily superior to other heuristic search techniques. Wolpert's no free lunch theorem [52] states that, "for any algorithm, any elevated performance over one class of problems is offset by performance over another class [of problem]". So, while it is likely that GAs can be tuned to perform better on train trajectory optimisation, this will be equally true for many other types of optimisation algorithm.

Finally, before GAs can be investigated for multi-train trajectory optimisation, a simulation methodology must be chosen for evaluating the performance of candidate control strategies. At the time this research was undertaken the author did not have access to a suitable multi-train simulator. The work presented by Yang et al. [44] used a GA to perform multi-train trajectory optimisation and was unusual because it did this in the context of a branched railway network. The algorithms used for the multi-train simulation were also documented in detail. For these reasons this thesis will use Yang's model as a starting point to explore multi-train trajectory planning.

Chapter 3

Multi-train trajectory optimisation to maximise rail network energy efficiency under travel-time constraints

This chapter is based on a published paper [53] [1] but includes additional detail on the implementation and validation of the model. The work presented focuses on trajectory optimisation of multiple trains in a network, with the aim of improving overall network punctuality and energy consumption. To do this, operational interaction between trains must be considered. For the reasons discussed in Section 2.4, the work by Yang et al. [44] was chosen as the most relevant state of the art and was therefore the starting point for further investigation. First, a new multi-train trajectory optimisation was

[1] Jonathan C J Goodwin, David I Fletcher and Robert F Harrison, Multi-train Trajectory Optimisation to Maximise Rail Network Energy Efficiency Under Travel-time Constraints, Proc IMechE Part F: J Rail and Rapid Transit, 230 (4), pp. 1318-1335.

implemented in C++, reproducing the state of the art. This is described in Section 3.1 along with the model's validation against published results and a number of improvements. Next, two specific limitations were described in Sections 3.2 and 3.3. Algorithms were proposed to address these and the performance of the resulting optimisations investigated in Section 3.4. Together, these improvements were shown to optimise an average of 27.6% further than published results in combination with increasing consistency of optimisation by a factor of 28 (Section 3.5.3). Finally, the improved optimisation consistency allowed a more detailed investigation into the effect of varying α, the weighting between different objectives in the cost function, in Section 3.5.

3.1 Implementation and validation of G1

The network N1 (illustrated in Figure 3.1) was previously investigated by Yang et al. [44] using the modelling methodology and optimisation hereafter referred to as Y1. Basic familiarity with Y1 is assumed, for which an overview can be found in Section 3.1.1. Y1 was implemented from scratch, and this new model is referred to as G1. Every effort has been made to ensure that G1 is as accurate a reproduction of Y1 as possible, enabling comparison with results from improved optimisation procedures G2, G3 and G4. All G1 experiments were tested on the same railway network as used in [44] and used the same procedures, constraints and variables unless otherwise stated.

Using control sequences resulting in indistinguishable trajectories, the simulation results from Y1 and G1 also gave the same total traverse energy and total traverse time to within 0.02% and 0.00% respectively. If links are compared individually then the variations in the energy and time are slightly larger, averaging ±0.25% and ±0.09% respectively, but cancelled due to the

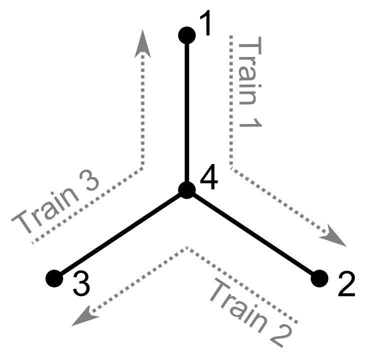


Figure 3.1: Illustration of network N1, the topology and train journeys of which were previously defined and investigated by Yang et al. [44]

cost function formulation. However, these errors are still relatively small and are most likely due to slightly different methods of calculating the braking point, and/or different rounding assumptions made at the start and end of braking (see Section 3.1.3).

The following points highlight important areas in the implementation of G1.

- The fuzzy variables mentioned by Yang were found to simplify analytically, allowing them to be implemented as conventional (single valued) variables.

- As will be justified below, in Section 3.1.2, it was necessary to use a simplified traction force of $f(v) = 360$ kN, over the whole range of velocities, for consistency with the output from Y1. This led to modelling a less realistic train but enabled comparison of optimisation results. As the primary aim of this research was to investigate trajectory optimisation techniques, the realism of the system being modelled was not critical.

- Also below, an algorithm is proposed for establishing the proper site to apply the braking operation, at the end of each link. It was not specified in Y1 how the braking site was calculated.

- On closer inspection of the algorithms described in [44] it can be seen that the energy calculation method in Y1 uses the resultant force (traction force + resistance force) on each train to calculate its traction power consumption. This has profound implications on the calculated energy use of trains suggesting that, in all situations, speed holding uses no energy. It is expected that using the traction force alone would prove to be a more realistic formulation. However, to allow comparison, these unmodified algorithms were used during validation of G1 but will be revised in Section 3.5.

3.1.1 Overview of Yang's formulation

Yang et al. [44] described a multi-train trajectory model and optimisation. This formulation will be referred to as Y1 throughout this thesis. Y1 modelled the rail network in Figure 3.1 as a finite graph; nodes representing stations, and edges representing bidirectional single-track railway links. Each link has a length, over which a speed limit profile is defined, whereas the nodes have no modelled properties. Unless stated otherwise, all optimisation investigated in this chapter were applied to the same network (N1).

In Y1, train motion on each link is defined as alternating sections of maximum traction and coasting, controlled by position vector $\underline{x}$, with application of the maximum braking operation interrupting the final coasting section at distance y to ensure stopping at the end of the link (Figure 3.2). The notation used in this chapter is listed in Appendix II and values of parameters are given in Table 3.1.

Train movements defined by each network control strategy, $\underline{\boldsymbol{X}}$, are simulated by implementing Newton's laws of motion using a piece-wise linear approximation ($\Delta t = 1$ s). Details of this are given in Appendix IV. Links are

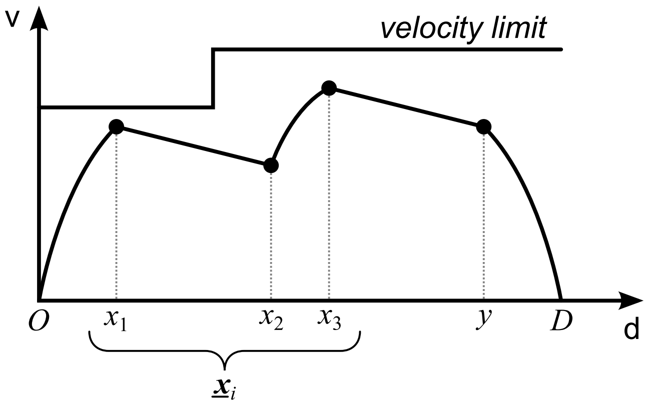


Figure 3.2: Traction and coasting operations are applied alternately as each link is traversed. The position of the control points determines the trajectory each train follows between origin O and destination D.

traversed in the order defined by $\underline{\boldsymbol{X}}$ and constraints imposed during simulation to ensure: feasible solutions, safe operation, ride comfort and sufficient time for operations at stations. As well as checking the feasibility of each solution, the simulation allows an objective function to be evaluated for each $\underline{\boldsymbol{X}}$. Target values for the total traverse time and total energy consumption are defined, as T and E respectively. The deviations from these targets were then formulated into a single equation, Equation (3.1), using a linear weighted sum method:

$$F_\alpha(\underline{\boldsymbol{X}}) = \alpha \cdot \max\left\{\frac{E(\underline{\boldsymbol{X}}) - \bar{E}}{\bar{E}}, 0\right\} + (1-\alpha) \cdot \max\left\{\frac{T(\underline{\boldsymbol{X}}) - \bar{T}}{\bar{T}}, 0\right\} \tag{3.1}$$

where the weighting factor, $\alpha \in [0,1]$, allows a different relative importance to be placed on energy consumption or traverse time.

Since Equation (3.1) only considers energy and time spent traversing links, a penalty accounting for operational interactions in stations, $G(\underline{\boldsymbol{X}})$, is added to Equation (3.1). This results in the overall objective score for each network control strategy, given in Equation (3.2). $G(\underline{\boldsymbol{X}})$ can be customised for different

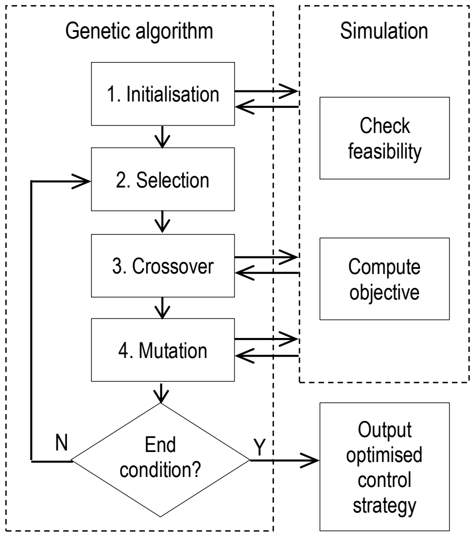


Figure 3.3: The structure of the optimisation algorithm used in Y1.

situations, but here is defined as the sum of departure delays, weighted by the relative priorities of different trains.

$$\text{Objective score} = F_\alpha(\underline{\boldsymbol{X}}) + G(\underline{\boldsymbol{X}}) \tag{3.2}$$

A GA (Figure 3.3) is used to minimise Equation (3.2), by searching for near-optimal $\underline{\boldsymbol{X}}$. Constraint checking is integrated into the genetic operators to ensure that any offspring, resulting from the breeding of parent chromosomes, is a feasible solution. The overall process is represented in Figure 3.3, where the loop will keep iterating new populations of solutions (expected to increase in fitness) until the end condition is reached. For Y1, a fixed number of generations is defined, after which the best solution found is accepted. For consistency, the same parameters as used in Yang's best optimisation (Table 3.1) are used when implementing G1 .

Table 3.1: The GA parameters used in this chapter [44, Table 4, experiment 9]. *Personal communication from L. Yang [54].

Parameter	Value
P_c (probability of crossover)	0.6
P_m (probability of mutation)	0.8
`pop_size`	40
Number of generations	800
α (weighting between energy and time)	0.3
M (initial mutation distance) *	100 m
Tr (minimum distance between operational transitions) *	500 m
Parameter used in roulette wheel selection (also referred to as α in [44]) *	0.2

3.1.2 Validating traction trajectories

In [44] the traction force (kN), of all trains on the network, is modelled using the piecewise function of train velocity given in Equation (3.3).

$$f(v) = \begin{cases} 360 & if 0 \leq v \leq 180 km/h \\ 360 - \frac{6}{7}(v - 180) & if 180 \leq v \leq 300 km/h \end{cases} \tag{3.3}$$

When Equation (3.3) was combined with the other forces, and the piecewise-linear model of train movement implemented in G1, the result appears to show identical traction characteristics to Y1 below about 200 km/h. However, as can be seen in Figure 3.5, above this velocity the trajectories diverge - with Y1 following instead the profile described by G1 using the traction force given by Equation (3.4).

$$f(v) = 360, if 0 \leq v \leq 300 km/h \tag{3.4}$$

Equation 3.3 is equivalent to an 18 MW train, which would already be one of the highest power high-speed trains in production, while Equation 3.4

is equivalent to a 30 MW train. This suggests that Equation 3.4 does not realistically model any existing train but has still been used in G1 to provide consistency with the traction characteristics demonstrated by Y1. As the primary aim of this research was to investigate trajectory optimisation techniques, the realism of the system being modeled was not critical. The traction characteristics resulting from Equations 3.3 and 3.4 are compared in Figure 3.4 and the resulting train acceleration profiles in Figure 3.5.

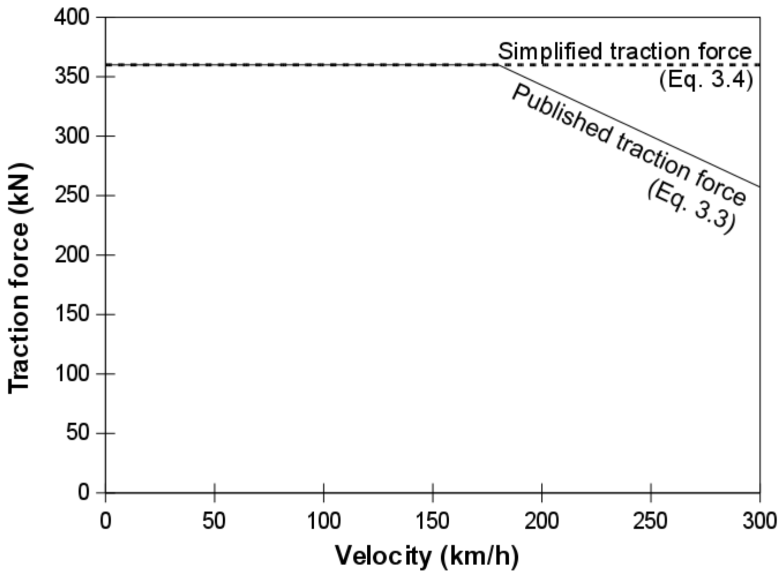


Figure 3.4: Comparison of traction characteristics.

3.1.3 Implementing braking

Yang mentions “a suitable brake site” and “the proper site for using braking operation” but in neither case specifies how this is found. Another relevant consideration not discussed in [44] is the rounding error introduced at the end of the link. In Y1 the train must not come to rest before the end of the link, as passing the end of the link is the end condition for the braking algorithm. This means any residual velocity at the end of the line will result in a faster traverse

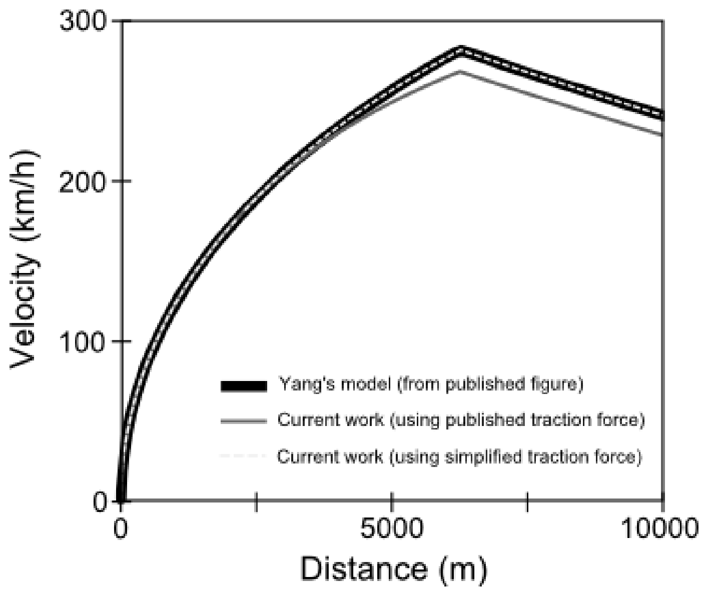


Figure 3.5: Comparison of simulated train velocity profiles resulting from different traction characteristics.

time but increased energy consumption compared to the ideal case. Given the discretised nature of the model, which only switches operational modes after an integer number of seconds, it is unlikely that braking will bring a train to rest at exactly the link end position. The proposed method of calculating the final coast-brake segment used in G1 also removes this small systematic error. G1 determines the correct site for braking implicitly by using a back calculated stopping process [32] [55] detailed in Procedure 1.

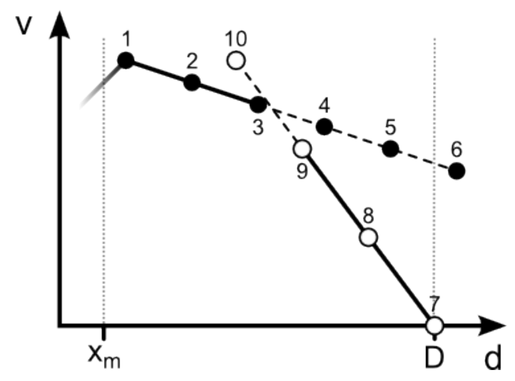


Figure 3.6: Finding the site for braking.

Procedure 1: Back calculated stopping process.

Step 1. Calculate the coasting profile to the end of the link in the normal way (Figure 3.6 - points 1 to 6).

Step 2. Calculate the braking profile backwards from the end of the link until it crosses the coasting profile (Figure 3.6 - points 7 to 10).

Step 3. The two profiles are then joined by averaging the velocity over the remaining distance (between points 3 and 9 on Figure 3.6), allowing an estimate of the time interval to be made. Energy consumption between points 3 and 9 was approximated to be half coasting and half braking.

3.1.4 Validating simulation

The train model in G1 was implemented assuming the linear traction force Equation (3.4) and back calculated stopping process (Procedure 1) detailed above. To estimate error in model G1, control points for the simulation were taken directly from [44, Fig. 5], instead of using the GA optimisation, and validated against the data for Y1 from [44, Table 5].

If the magnitudes of the errors are summed, then the total difference in

Table 3.2: Comparison of simulation outputs using the same control strategy.

	Y1		G1		G1 error (%)	
Train (journey)	Energy /kWh	Time /s	Energy /kWh	Time /s	Energy	Time
1 (1 to 4)	798.015	664	799.725	663.4	0.21	-0.09
1 (4 to 2)	763.768	724	761.101	724.1	-0.35	0.02
2 (2 to 4)	884.459	663	884.852	664.6	0.04	0.25
2 (4 to 3)	881.108	644	881.532	643.9	0.05	-0.01
3 (3 to 4)	805.627	639	808.383	638.3	0.34	-0.10
3 (4 to 1)	768.395	653	764.676	652.5	-0.48	-0.07
Network total	4901.371	3987	4900.269	3986.9	-0.02	0.00

energies and times for each journey is 1.48 kW h and 0.54 s respectively over the whole network. These errors are negligible when one considers that the total distance travelled by trains in the model is 180 km. However, since the error in G1 does not appear to be systematically high or low, a simple summation of errors cancel to -0.18 kW h and -0.02 s. It is these values which are used by the cost function in Equation (3.1) to score the network, meaning that both Y1 and G1 have the same score of 0.0331. This is the measure ultimately used by the optimisation process to compare the fitness of different control strategies, meaning Y1 and G1 are indistinguishable (to 3 s.f.) for the simulated control strategy.

Overall, it appears that the reproduced model is a good representation of the original, with the nominal differences probably arising from alternative methods for calculating the braking point for trains at the end of each link.

3.1.5 Validating optimisation performance

As with the model/simulation presented in Section 3.1.4, every effort has been made to implement the GA in G1 using the same algorithms and parameters described for Y1. [44] Table 3.1 lists the parameter values used, some of which were obtained via personal communication [54]: It was also assumed that each link was initialised to have 3 switching points. Notation used to describe both Y1 and G1 can be found in Appendix II.

For many of the parameters used in Y1 there was little justification given for the values used. However, the authors did document a simple optimisation of the following GA parameters:

- probability of performing a crossover operation (P_c)
- probability of performing a mutation operation (P_m)
- total number of chromosomes in the population (`pop_size`)
- number of generations for which the GA was run

Ten different sets of these parameters were randomly generated by Yang, and the optimisation run just once with each set. This means the consistency of Y1 (using the same set of parameters) was not recorded, but only the variance in the optimised score (using different sets of parameters). This variance, of 0.0001, was stated as illustrating "the robustness and steadiness of the proposed solution methodology", even though it is equivalent to a standard deviation of 0.012, or $\pm 25\%$ of the average score. Yang then selected the set of parameters used in finding the best optimised score, and used them in some further investigations. The same sets of parameters tested by Yang were adopted for the optimisation in G1, with each optimisation being run one hundred times ($n = 100$) to assess the consistency of the optimisation process (see Table 3.3).

The consistency of the optimisation in G1 appears to be quite poor, with each set of parameters tested having a large standard deviation compared to its average score. Also, the average and standard deviation in scores varies

Table 3.3: Optimisation results using G1 with the sets of parameters from [44, Table 4]. The optimisation was run one hundred times ($n = 100$) with each set of parameters. *Scored using Equation (3.2) - lower is better.

Set of Parameters	P_c	P_m	`pop_size`	Generations	Mean computation time /s	Mean score*	σ in score
1	0.7	0.8	40	1000	116	0.052	0.018
2	0.4	0.5	30	1000	59	0.055	0.018
3	0.4	0.6	30	800	52	0.053	0.016
4	0.5	0.7	30	900	61	0.050	0.019
5	0.6	0.7	40	1000	96	0.051	0.017
6	0.5	0.6	30	1500	87	0.051	0.018
7	0.8	0.5	30	1000	57	0.054	0.019
8	0.8	0.9	40	1000	120	0.051	0.017
9	0.6	0.8	40	800	82	0.055	0.019
10	0.7	0.9	50	800	112	0.048	0.015

Table 3.4: Combined optimisation results. *Scored using Equation (3.2) - lower is better.

Experiment	n	Mean computation time /s	Mean score*	σ in score
G1 combined	1000	84	0.052	0.018
Y1 combined	10	775	0.048	0.012

only slightly between different sets of parameters, which suggests that varying these parameters has little effect on the optimisation. This allows all the repeats to be combined into one larger dataset even though they are using different sets of parameters. The result of this, for both Y1 and G1, can be found in Table 3.4.

The two larger data sets, Y1 combined and G1 combined, allow the distributions of optimisation results observed in Y1 and G1 to be compared. However, given a sample of their results it cannot be proven that the two models are identical. Instead, the probability the null hypothesis (that Y1 and G1 are identical) is incorrect can be estimated, allowing it to be rejected with a given

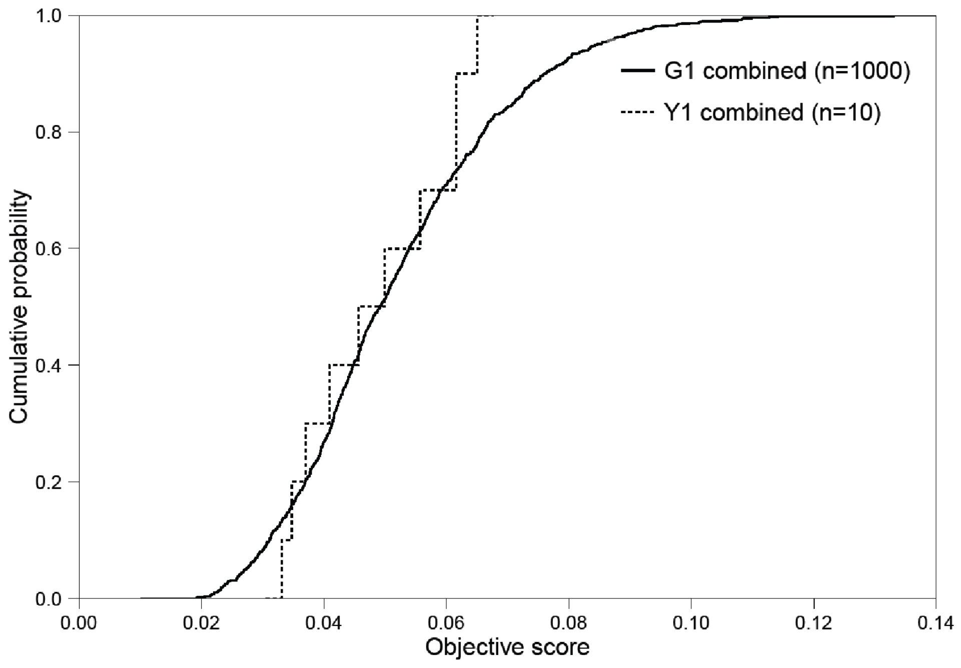


Figure 3.7: Empirical cumulative probability distributions of optimisation results.

degree of confidence. A p-value is the probability that the null hypothesis is rejected even when it is true. The Kolmogorov-Smirnov test was carried out using R [56] to compare the two distributions in Figure 3.7. This yielded a p-value of 0.7293, meaning that the null hypothesis cannot be rejected using the typical confidence level of 0.05. In-fact, even using the extremely relaxed confidence level of 0.7 the null hypothesis can still not be rejected. In other words there is no significant difference between Y1 and G1.

Simulations were run on one core of a personal computer with an AMD Phenom II N850 Triple-Core 2.2 GHz processor. It is therefore surprising that the average computational time per optimisation was only 11% of that reported by Yang, who used a higher specification 2.67 GHz Intel processor. This difference in optimisation speed may reflect the efficiency with which the algorithms were implemented or simply the level of optimisation used when compiling the code. The larger variation in the optimisation quality of G1 and the small number of published results from Y1 makes agreement between the optimisation procedures difficult to prove with a high degree of certainty. However, there is very little evidence that optimisations by Y1 and G1 give significantly different distributions of results.

3.1.6 Method of traction energy calculation

On closer inspection of the algorithms in Yang et al. [44] it was found that the traction energy consumption was calculated using the resultant force acting on each train (Δwork = resultant_force $\times$ Δdistance) using a piece-wise linear approximation. This formulation meant that increased resistance forces at high speed caused a reduction in resultant force and, therefore, a reduction in the energy use of trains. To enable like-for-like comparison with previously published results, the initial investigation into the performance of different optimisations was performed without changing the method of energy calculation (Section 3.4). However, the more realistic formulation of calculating energy using (Δwork = traction_force $\times$ Δdistance) was adopted for Section 3.5.2 and all subsequent investigations.

3.2 G2: Link-wise mutation operation

The mutation operation proposed by Yang has the advantage that it tends towards the previous solution, which is known to be feasible. However, this places extra constraints on the optimisation process; in this case requiring the same mutation size of all control points on the network. Below, a modified mutation operation is proposed that finds separate feasible mutation sizes for each link independently. This requires the ability to alternately apply a genetic operator to, and then check the feasibility of, the control strategy for each link in the network. A genetic operator that is applied in this way will be called a *link-wise* genetic operator and will be applied using Procedure 2 (detailed below). A mutation operation adapted to work as a link-wise operator is proposed in Procedure 3. Together these procedures allow link-wise mutation to be performed on a population. It is intended that this should place fewer constraints on the optimisation process, thereby allowing better local optimisation.

Procedure 2: Alternating a genetic operation and feasibility checking.

```
Step 1.   For each chromosome (in any order)
Step 2.   If $P_h <$ rand$[0,1]$ then go to step 10
Step 3.   For each link control strategy (in the order) defined by $\underline{\boldsymbol{X}}$
Step 4.   Apply link-wise genetic operator ($\underline{x}'' = h(\underline{x}')$)
Step 5.   If $\underline{x}''$ is feasible then go to step 8
Step 6.   If $\underline{x}'$ is feasible then $\underline{x}'' = \underline{x}'$ and go to step 8
Step 7.   Else, go to step 10
Step 8.   Next link
Step 9.   $\underline{\boldsymbol{X}}''$ replaces $\underline{\boldsymbol{X}}'$ in the population
Step 10.  Next chromosome
```

Here P_h is the probability of applying the link-wise operator $h(\underline{x})$.

Procedure 3: Single link mutation.

```
Step 1.  Predetermine an initial distance of mutation $M > 0$, let $m = M$
Step 2.  Randomly give a mutation vector $\underline{d}$ with the same length as $\underline{x}'$
Step 3.  Let $\underline{x}'' = \underline{x}' + m\underline{d}$
Step 4.  Correct $\underline{x}''$ to the feasible form (using the procedure in Yang et al. [44])
Step 5.  Check validity of $\underline{x}''$ using simulation
Step 6.  If $\underline{x}''$ is feasible then end procedure, else let $m = m/2$
Step 7.  If $m >$ (a small positive distance) then go to step 3, else end procedure
```

Here $\underline{d}$ is a vector with elements randomly defined as +1 or -1

The mutation operation in G1 was replaced with the link-wise mutation operation (defined in Procedure 2 and Procedure 3) to make optimisation G2. Unlike G1, mutation in G2 does not guarantee that a feasible network control strategy will be produced. This is the same situation as already existed for the crossover operation. In the case where neither the mutated ($\underline{x}''$) or pre-mutation ($\underline{x}'$) link control sequences are feasible, Procedure 2 will reach step 7 and the current chromosome will not be mutated. However, the improvement in optimisation performance discussed later suggests that, in the system studied, the potential for this event to occur is outweighed by the benefit of having a less-constrained genetic operator.

3.3 G3: Insertion and deletion operations

As well as having good local optimisation, the other main problem that must be overcome in complex optimisation problems is how to avoid getting stuck in local minima. GAs seek to do this by having diversity within a population and also the potential to reintroduce lost diversity using mutation. However, as

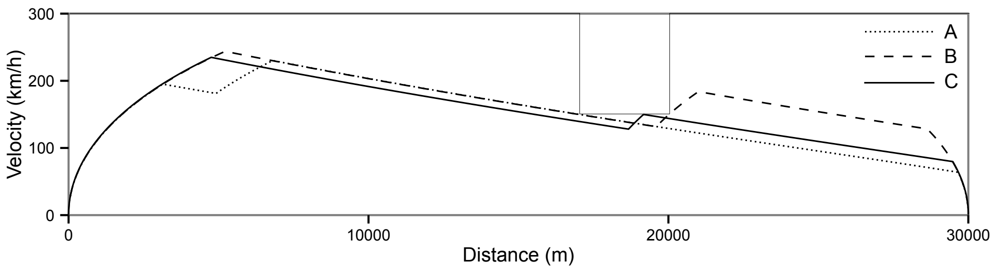


Figure 3.8: Train trajectories generated by optimisation G2. The lines A and B illustrate proposed local minima observed in the results. Using mutation, these must interconvert by passing through some unfavourable intermediate similar to the one illustrated by line C.

will be discussed in the results section, neither optimisation with the original mutation operation (G1) nor the proposed link-wise mutation operation (G2) appear to be successful in avoiding local minima. In particular, solutions with two distinct patterns of control strategies were observed: those with the second traction operation before the drop in line speed limit, and those with the second traction operation after the drop in line speed limit. These are illustrated in Figure 3.8 as A and B respectively. If the population has converged, and only contains one of these control strategy patterns, then the other can only be reintroduced using mutation. However, since the distance of reduced line speed limit (3 km) is large compared with the mutation size ($\leq$ 100 m), many generations of poorly scoring intermediate strategies make rediscovery of an A-like solution from a population of B-like solutions unlikely (and vice versa). If control points are excluded from a region of the line then, by definition, the mode of train control in this region cannot be changed, which may lead to a suboptimal solution. In this case, solution A fails to exploit the rise in line speed from 20,000 m onwards. Conversely, too many control points in a region may lead to a restricted control strategy, as a minimum distance between operation transitions must be maintained, again leading to suboptimal solutions.

It is probable that increasing the population size would cause diversity resulting in a reduced likelihood of getting stuck in local minima, but this would also greatly increase the computational burden from simulation. In biology there are three classes of single nucleotide mutation: point mutation, insertion, and deletion. Both the original mutation procedure and the link-wise mutation used in G2 are analogous to a DNA point mutation in biology, as one control point is modified, but the total number of control points remains the same. For this reason procedures are proposed for the probabilistic insertion and deletion of pairs of control points (see Figure 3.9). Chang and Sim [25] used similar operations, duplication and deletion, but it is believed the operations proposed here are more effective for the following reasons:

- The probability of insertions and deletions is biased towards locations where they are most likely to be needed.
- As much as possible, the effect of the operations on the 'downstream' trajectory is minimised, decreasing the probability of producing infeasible solutions.

Procedures 4 and 5 (detailed below) enable both of these and capture the following logic. It is proposed that the probability of insertion between two adjacent control points is proportional to the distance between them. This will bias insertion towards areas of the control sequence currently lacking control points. The total probability of insertion or deletion happening on each link was implemented as $P_{ins\_link} = 0.25$ and $P_{del\_link} = 0.25$, respectively (these probabilities were tuned 'by hand' and found to be large enough to give sufficient exploration, but small enough not to impede convergence). Using the notation illustrated in Figure 3.2 the probability of inserting a pair of control points between control points n and $n + 1$ is given by:

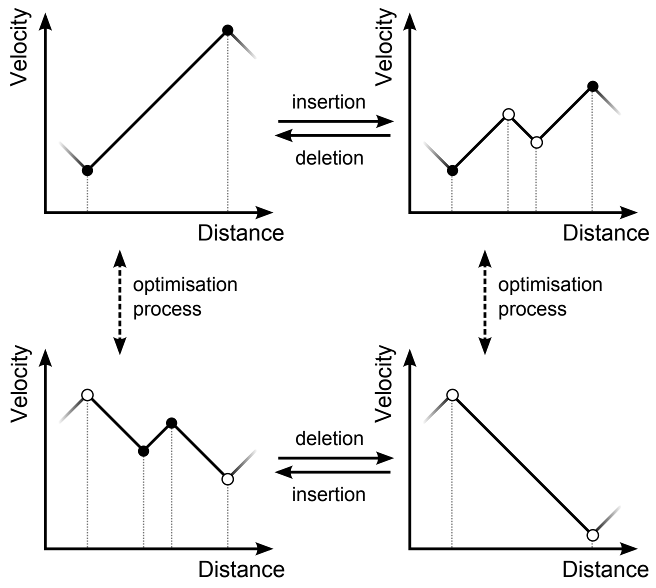


Figure 3.9: Extracts from train trajectories illustrating how they are affected by the insertion and deletion operations (for simplicity, modification of the neighbouring control points has not been shown here). During the optimisation process, control points may be moved by mutation and crossover, extending or contracting the distance for which the traction or coasting operation is applied.

$$P_{ins\_pair}(x_n) = P_{ins\_link} \frac{x_{n+1} - x_n}{D} \tag{3.5}$$

where $0 \leq n \leq n\_max$.

Similarly, the probability of deleting a pair of control points should be proportional to their 'shortness', to bias for removal of potentially redundant genetic material. The probability of deleting the pair of control points $n+1$ is given by:

$$P_{del\_pair}(x_n) = P_{del\_link} \frac{(1 - (x_{n+1} - x_n)/(x_{n\_max} - x_1))}{(n\_max - 2)} \tag{3.6}$$

where $1 \leq n \leq (n\_max - 1)$.

As can be seen in Figure 3.9, the insertion or deletion of control point pairs causes downstream changes to the velocity profile of the train. To limit this, and so maximise the chance of insertion or deletion resulting in a feasible solution, two strategies are proposed. The first is to minimise the distance between the inserted pair of control points (i.e. $\Delta d = Tr$, the minimum distance between operational transitions). The second is to move the position of neighbouring control points in order to conserve the total distance over which each control operation is applied. As with link-wise mutation, the insertion and deletion procedures were applied probabilistically to the population using Procedure 2 (step 4), with a probability of $P_i = 0.6$ and $P_d = 0.6$, respectively (again, these were tuned 'by hand' in combination with P_{ins_link} and P_{del_link}).

Procedure 4: Link-wise insertion (valid for $n\_max \geq 1$)

```
Step 1.   Let $n = 0$
Step 2.   If $P_{ins\_pair}(x_n) < \texttt{rand}[0,1]$ then go to step 14
Step 3.   If $(x_{n+1} - x_n) < 2Tr$ then go to step 11
Step 4.   If $0.5 < \texttt{rand}[0,1]$ then go to step 8
Step 5.   If $n = 0$ then go to step 9
Step 6.   If $(x_n - x_{n-1}) < 2Tr$ then go to step 11
Step 7.   $x_n \leftarrow x_n - Tr$, go to step 11
Step 8.   If $n = n\_max$ then go to step 6
Step 9.   If $(x_{n+2} - x_{n+1}) < 2Tr$ then go to step 11
Step 10.  $x_{n+1} \leftarrow x_{n+1} + Tr$
Step 11.  If $(x_{n+1} - x_n) < 3Tr$ then go to step 14
Step 12.  Let $d = x_n + Tr + ((x_{n+1} - xn) - 3Tr) \cdot \texttt{rand}[0,1]$
Step 13.  Insert new control points into $\underline{x}$ at position $d$ and $d + Tr$
Step 14.  $n \leftarrow n + 1$
Step 15.  If $n \leq n\_max$ go to step 2
```

Table 3.5: Summary of the major innovations of each optimisation procedure defined in this paper.

Optimisation	Innovation
G1	Implementation of the model and GA optimisation described by Yang et al. [44]
G2	Introduces a new (link-wise) mutation operation to replace the original mutation operation of G1
G3	Introduces the new genetic operations of insertion and deletion alongside the original GA optimisation of G1
G4	Combines the innovations of G2 and G3

Procedure 5: Link-wise insertion (valid for $n\_max \geq 1$)

```
Step 1.   Let $n = 1$
Step 2.   If $P_{del\_pair}(x_n) < \texttt{rand}[0,1]$ then go to step 10
Step 3.   Let $d = x_{n+1} - x_n$
Step 4.   If $0.5 < \texttt{rand}[0,1]$ then go to step 7
Step 5.   If $n = 1$ then go to step 8
Step 6.   $x_{n-1} \leftarrow x_{n-1} + d$, go to step 9
Step 7.   If $n = (n\_max - 1)$ then go to step 6
Step 8.   $x_{n+2} \leftarrow x_{n+2} - d$
Step 9.   Remove control points $x_n$ and $x_{n+1}$
Step 10.  $n \leftarrow n + 1$
Step 11.  If $n \leq (n\_max - 1)$ go to step 2
```

G3 was implemented by adding the insertion in deletion operations to G1. A summary of the major innovations of each optimisation procedure defined in this chapter is presented in Table 3.5.

3.4 Comparing optimisation performance of G1 to G4

For each of the formulations described in Table 3.5, 100 independent optimisations were carried out to assess the effectiveness and consistency of the optimisation process. Initialisation of 100 populations was also performed, without any further optimisation, and the best solution from each population recorded. Comparison of these results is given in Table 3.6 and Figure 3.10 followed by a detailed analysis of each individual optimisation. The optimisation dynamics of G1 and G4 are also compared. Ideally an optimisation would consistently find the solution that has the lowest score (i.e. the global optimal solution); so the smaller the spread in scores, and the lower the scores found, the better the optimisation. However, since the objective score associated with the globally optimal solution is not known for this system, the performance of each optimisation is quantified relative to the performance of G1 using Equation 3.7 and Equation 3.8.

$$\text{\% improvement in mean score achieved by GX} = 100 \times \frac{S_{GX} - S_{G1}}{S_{G1} - S_{G0}} \tag{3.7}$$

$$\sigma \text{ ratio (fractional improvement in consistency) achieved by GX} = \frac{\sigma_{G1}}{\sigma_{GX}} \tag{3.8}$$

where G0 is no optimisation (random initialisation only), GX is any optimisation (G1 to G4), S_{GX} is the mean score after optimisation with GX, and σ_{GX} is the standard deviation in objective scores after optimisation with GX.

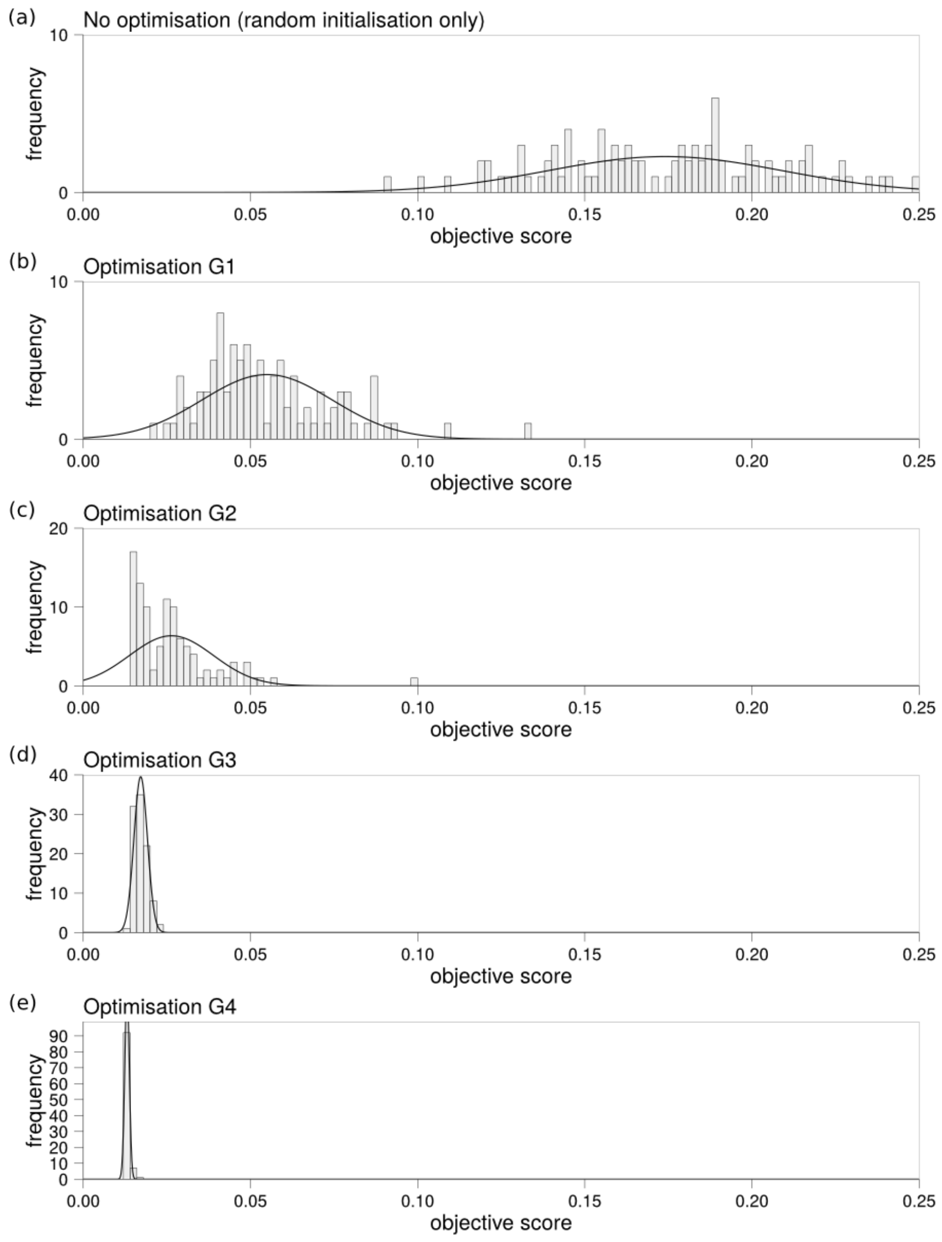


Figure 3.10: Histograms comparing the distribution of results from different optimisation techniques (lower scores are better). The improvement in optimisation performance from (a) to (e) can be seen by the monotonic decrease in the mean and standard deviation in scores achieved. Normal distribution curves are shown for clarity, although strictly only the data in (a) is normally distributed having a (Shapiro-Wilk) p-value > 0.05. [56] The significance of the multimodal distribution observed in (c) is discussed below.

Table 3.6: The result of optimisation using G1 to G4 (each assessed using a sample of 100 independent optimisations).

	Objective score		Improvement compared to G1	
Optimisation	Mean	σ	Mean (%)	σ ratio
None (random initialisation only)	0.1740	0.0349	-100.0	0.6
G1	0.0550	0.0195	0.0	1.0
G2	0.0264	0.0126	24.0	1.6
G3	0.0172	0.0020	31.8	9.7
G4	0.0131	0.0007	35.2	29.5

Optimisation using G1

It can be seen by comparing Figure 3.10(a) and (b) that G1 is effective in optimising the system described by Yang et al. [44]. However, after optimisation there is still a large variation in the objective score of results, caused by the trajectories of the optimised results that are illustrated in Figure 3.11. The trajectories show that in some places there is good consensus in the position of control points found (e.g. point A in part 3 of the Figure), whereas in other places (e.g. points B and C) large variations are clear. Large variation within a single, uninterrupted region of the search space is consistent with either poor local optimisation or lack of selection pressure where there is no significant change in objective score between different solutions. However, the large variation in objective scores seen in Figure 3.10(b) suggests the latter is unlikely. Also, as will be seen for optimisation with G2, if local optimisation is improved then C separates into two local minima. These issues are addressed by the innovations introduced in optimisation G2 and G3, respectively.

Optimisation using G2

The optimised profiles in Figure 3.10(c) have lower objective score values than in Figure 3.10(a) or (b) (i.e. better), but no longer appear to be normally distributed and instead a clustering of the results is observed. This suggests that G2 is finding local minima in the search space and is consistent with improved local optimisation. Both these inferences are supported by analysing the trajectories underlying the distribution of scores, shown in Figure 3.12. The improvement in local optimisation can be seen for most control points, specifically, the variation in positions found for control point B is much less than in Figure 3.11. Also, solutions place control point C (the position of the second traction application) in one of two well-separated locations. These two types of solution are not easily interconverted using the original mutation alone, so if one is lost from the population the search may become confined to a local minimum (see Figure 3.8).

Optimisation using G3

Optimisation G3 was specifically developed to address the occurrence of local minima in the optimised solutions, highlighted in the results of optimisation G2. It is clear from Figure 3.13 that this has been successful and that the trajectories of solutions found by G3 have a much clearer consensus. Figure 3.10(d) also shows that the objective scores resulting from these trajectories have a smaller variance and better average. It is particularly interesting to note that the optimised trajectory of train 3 (station 3 to 4) in Figure 3.13 now appears to approximate to the optimal profile we expect for a train on flat track: maximum traction, speed holding, coasting, and maximum braking. [11] However, a slight blurring of some trajectories in Figure 3.13 compared with the equivalent positions in Figure 3.12 suggests that G2 achieved slightly better local optimisation than G3.

Optimisation using G4

Optimisation G4 combines the innovations of G2 and G3 allowing it to find solutions with both a clear consensus and very little local variation in trajectories (Figure 3.14). Figure 3.10(e) also shows the improved optimisation performance and consistency. Together these give us much greater confidence that each optimisation using G4 will find a 'near optimal' network solution.

Optimisation dynamics

As well as different final results, the optimisations G1 to G4 also displayed different dynamics during the optimisation process. Figure 3.15 shows that after 800 generations there was still widespread variation among the G1 runs, whereas G4 runs consistently converged after about 200 generations.

3.5 Investigating system properties

3.5.1 Trade-off between energy consumption and traverse time

When scoring each network control strategy, $\underline{\boldsymbol{X}}$, both G1 and G4 use Equation (3.1) to determine the contribution of energy and time to the objective score. There is a region of the search space, $E(\underline{\boldsymbol{X}}) \geq \bar{E}$ and $T(\underline{\boldsymbol{X}}) \geq \bar{T}$, where $\underline{\boldsymbol{X}}$ does not meet either the energy or the time target. Most solutions are expected to lie in this region since, in general, going faster uses more energy and there is no improvement in score once the targets have been achieved. In this region Equation (3.1) reduces to:

$$T(\underline{\boldsymbol{X}}) = m_\alpha \cdot E(\underline{\boldsymbol{X}}) + c_\alpha \tag{3.9}$$

where $m_\alpha = -\frac{\alpha \bar{T}}{(1-\alpha)\bar{E}}$, and $c_\alpha = \frac{\bar{T}(F_\alpha(\underline{\boldsymbol{X}})+1)}{(1-\alpha)}$

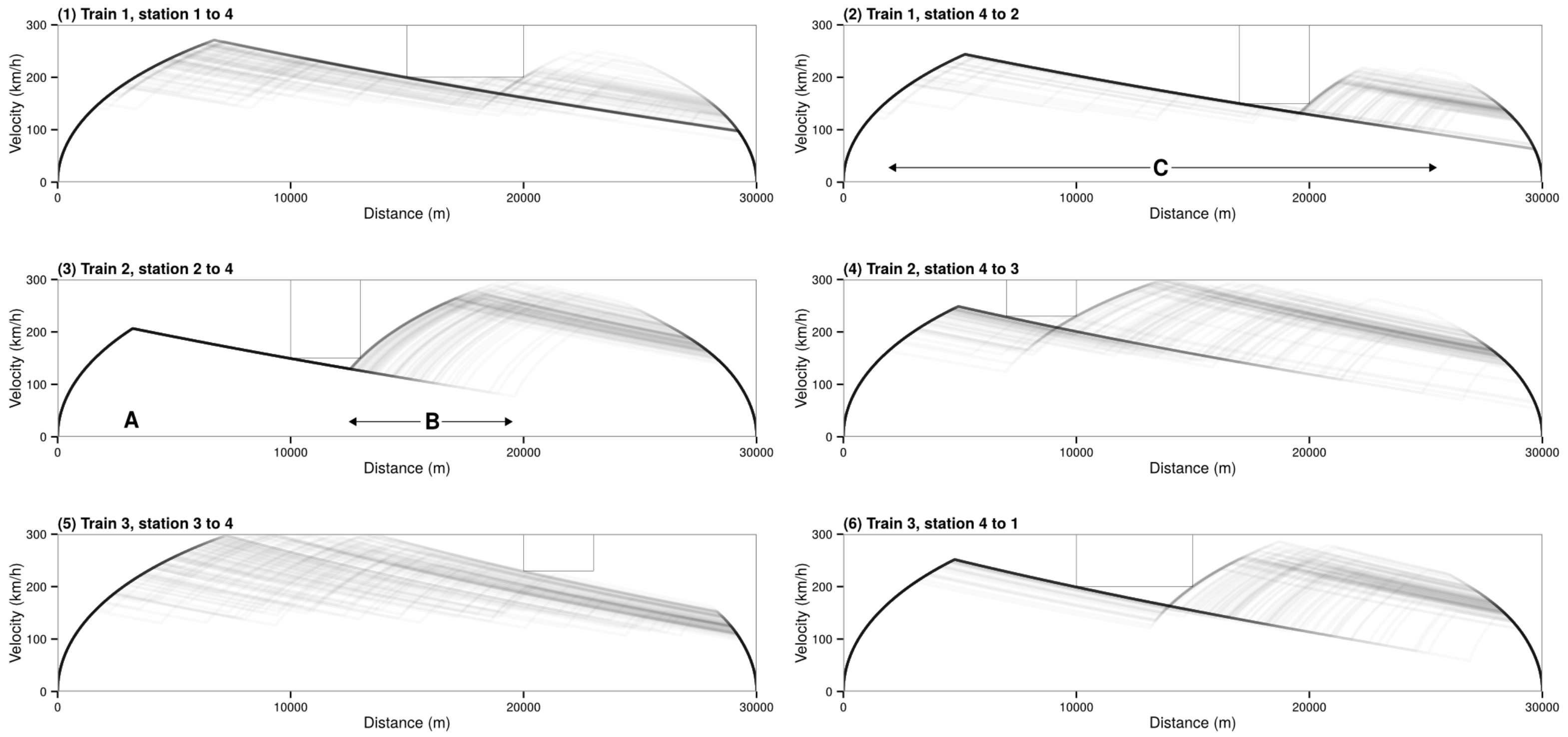


Figure 3.11: The consistency of train trajectories found using G1 to optimise N1 (100 independently optimised trajectories are overlaid). The position of control points are labelled to indicate: A - strong consensus, B - large local variation, C - near global variation.

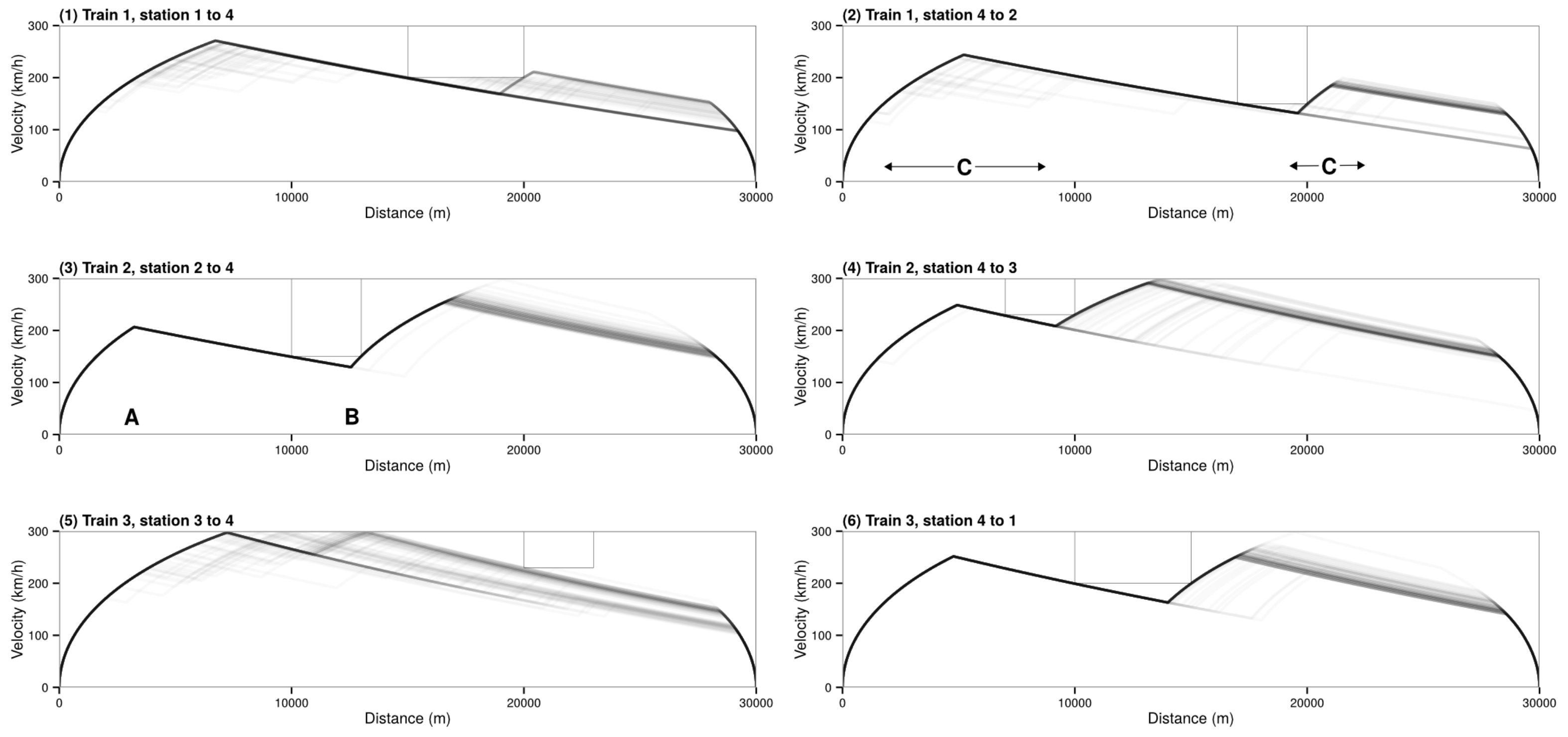


Figure 3.12: The consistency of train trajectories found using G2 to optimise N1 (100 independently optimised trajectories are overlaid). There is a strong consensus in the position of control points A and B; however, the two distinct locations of C suggest at least two different local minima are present in optimised solutions.

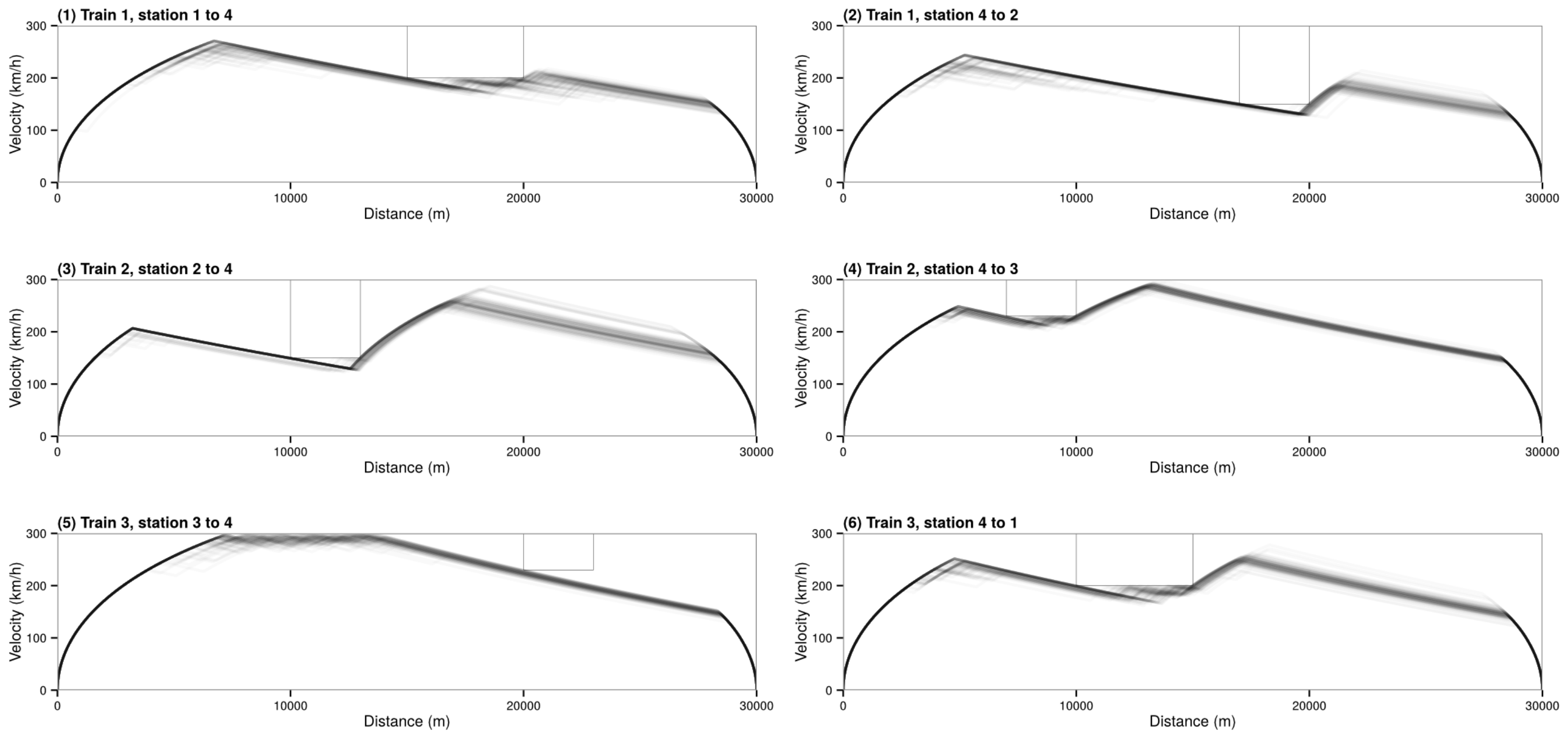


Figure 3.13: The consistency of train trajectories found using G3 to optimise N1 (100 independently optimised trajectories are overlaid). A clear consensus is seen, though blurring of some trajectories suggests there is a small amount of local variation.

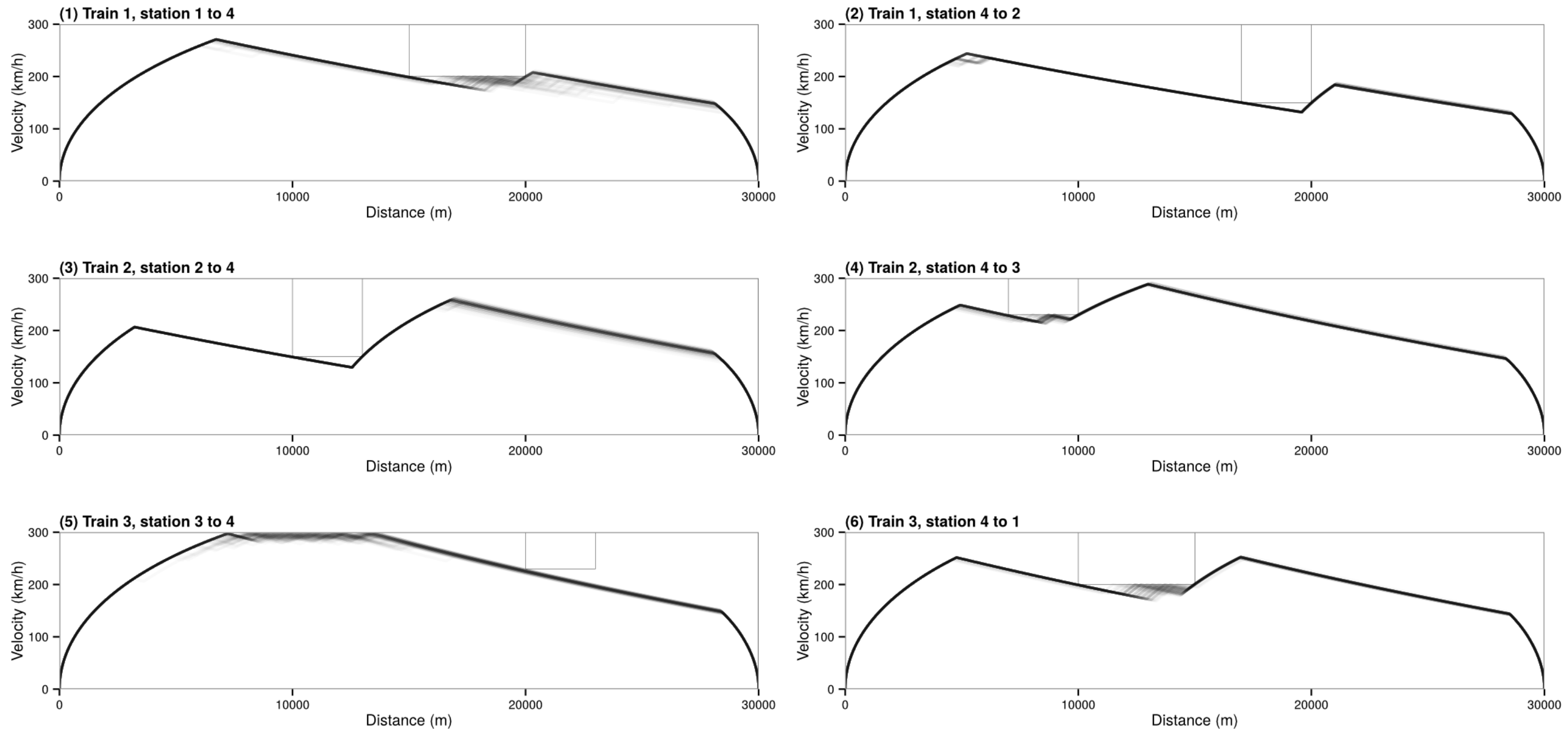


Figure 3.14: The consistency of train trajectories found using G4 to optimise N1 (100 independently optimised trajectories are overlaid). A clear consensus is observed along with minimal local variation on trajectories.

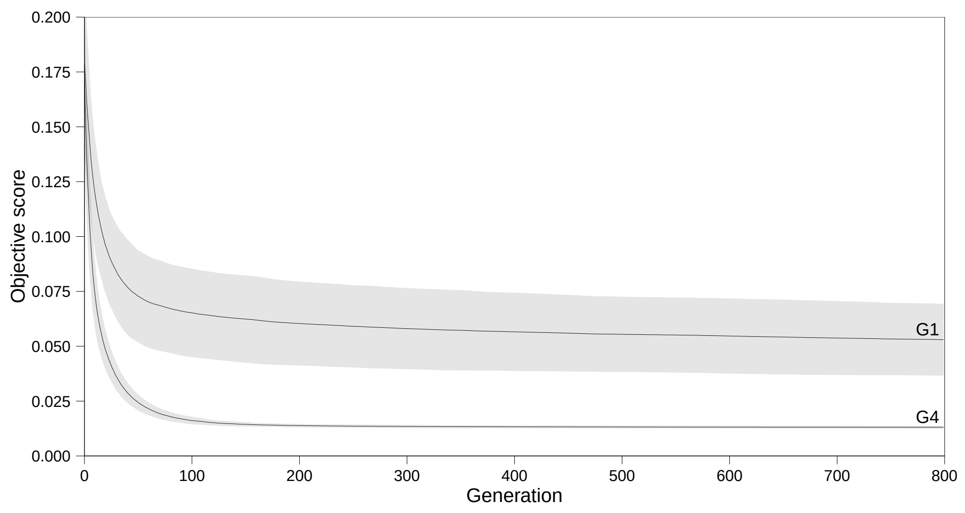


Figure 3.15: Average genetic algorithm progress from 100 optimisations using G1 and G4. The grey areas show the one standard deviation about the mean objective score levels.

This defines a line of constant $F_\alpha(\underline{\boldsymbol{X}})$ along which the combinations of energy and time are equivalent in the cost function. For the above investigations using G1 to G4, $\alpha = 0.3$, $\bar{E} = 4800$ kWh, and $\bar{T} = 3840$ s, so the gradient of this line is, $m_\alpha = -0.3429$ (this will vary with the parameters chosen). The intercept c_α is dependent on the level of optimisation. In solutions from G4 the penalty for delays $D(\underline{\boldsymbol{X}})$ is usually very small (mean = 0.0003, standard deviation = 0.0005), so we can assume that the objective score $\sim F_\alpha(\underline{\boldsymbol{X}})$. The lowest G4 score of 0.0131 gives an intercept, $c_\alpha \sim 5558$. This line of best score is shown on Figure 3.16, along with the energy and times of solutions obtained using different methods. In optimisations with two or more competing objectives there often exists a set of solutions where one objective can not be improved without increasing a different objective. Such solutions are said to be Pareto optimal and the set of these solutions makes up the Pareto front. The line of best score, calculated using Equation (3.1), appears give a tangent

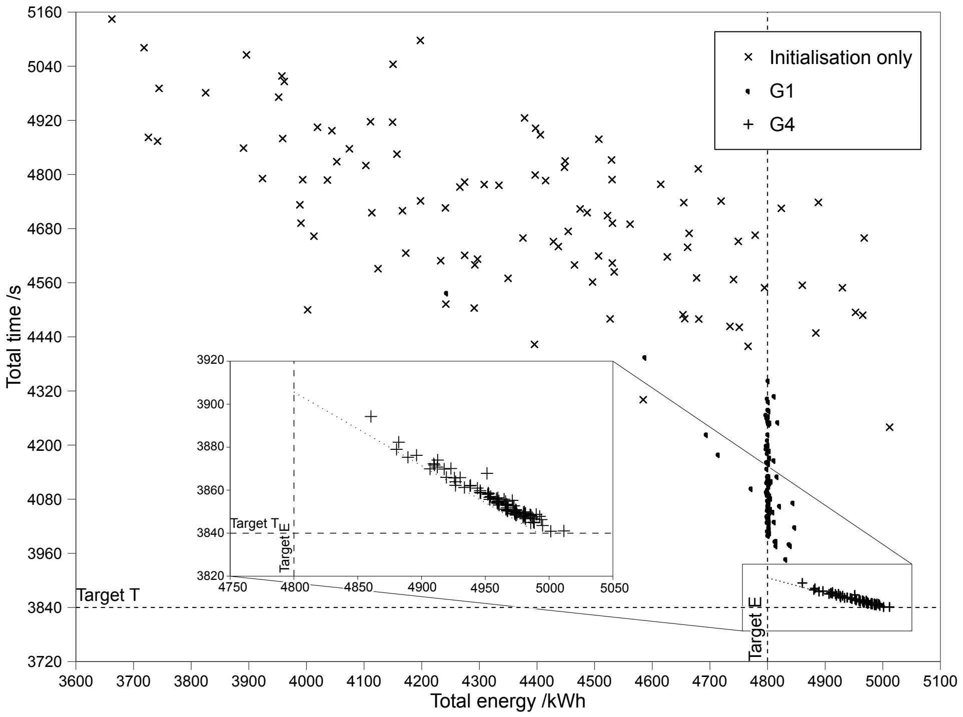


Figure 3.16: The total energy and traverse times of the network solutions obtained by: random initialisation only, optimisation using G1 and G4 (100 solutions of each). Equation (3.9) is used to find the line of constant $F_\alpha(\underline{\boldsymbol{X}})$ for the best scoring solution found by G4 (dotted line). It can be seen from the expanded area that G4 solutions vary in energy and time, but all have very similar objective scores.

to the Pareto front in Figure 3.16.

It is clear from Figure 3.16 that both optimisations lead to better solutions when compared with the randomly generated initial solutions. However, G1 solutions appear to be clustered around the target energy limit but with a large variation in total time, leading to a large variation in score. In contrast, all the G4 solutions are located close to the line of constant $F_\alpha(\underline{\boldsymbol{X}})$, again suggesting that it is a much better and more consistent optimisation. It can also be seen that some solutions found by G4 meet the target time, whereas others are much closer to meeting the target energy. It seems likely that the trajectories found using lower α and therefore placing a higher importance on target time, would not be significantly different from the solutions found with $\alpha = 0.3$ and that increasing α may also have little effect. For this reason, before investigating the effects of varying α, a new method of traction energy calculation is introduced. Not only is this method based on a more realistic formulation, but by increasing the energy consumption at high speeds it also increases the difference between solutions that can achieve the target energy consumption and target traverse time.

Revised method of energy calculation

From this point onwards the formulations of G1 to G4 have all been amended to use the more realistic method of traction energy calculation described in Section 3.1.6. With this improved formulation the optimisation G4 now yields trajectories that appear to exhibit an approximation to speed holding at around 200 km/h, see Figure 3.17.

3.5.2 Effect of varying α

The weighting parameter $\alpha \in [0, 1]$ in Equation (3.1) can be varied. A low value of α means the optimisations will prioritise meeting the time target, whereas a high α will prioritise meeting the energy target. By varying α used in the scoring of optimisation G4 (Figure 3.18) we can see that the optimised objective scores appear to be proportional to α below $\alpha = 0.2$ (low α), and also above $\alpha = 0.4$ (high α). This is consistent with the total time and total energy of solutions being near constant in this region, which Figure 3.19, showing the output of multiple repeated simulation runs, confirms to be the case.

Figure 3.19 appears to show a Pareto front similar to those typically found when comparing run times versus energy consumptions of single-train opti-

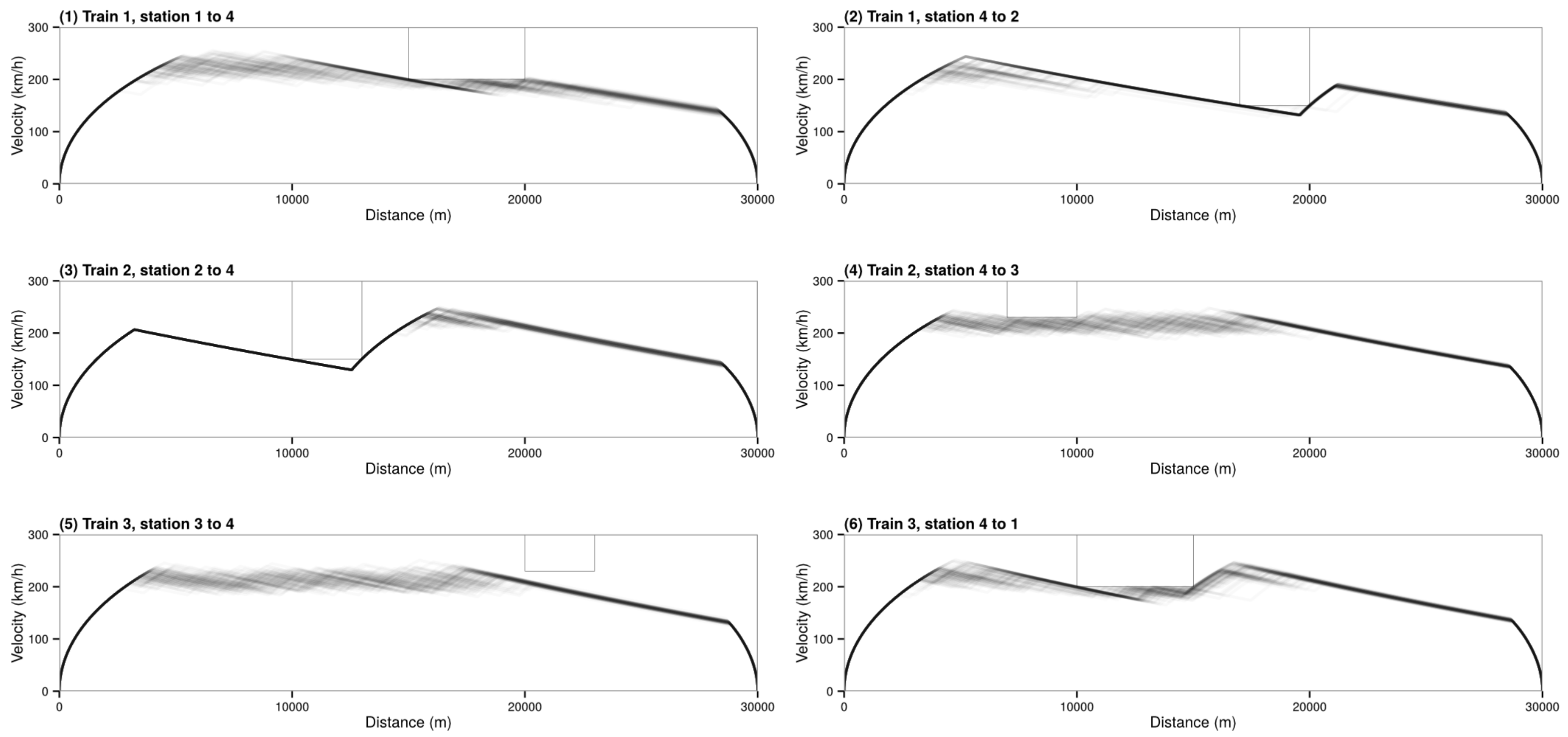


Figure 3.17: The consistency of train trajectories found using the new formulation of G4 to optimise N1 (100 independently optimised trajectories are overlaid).

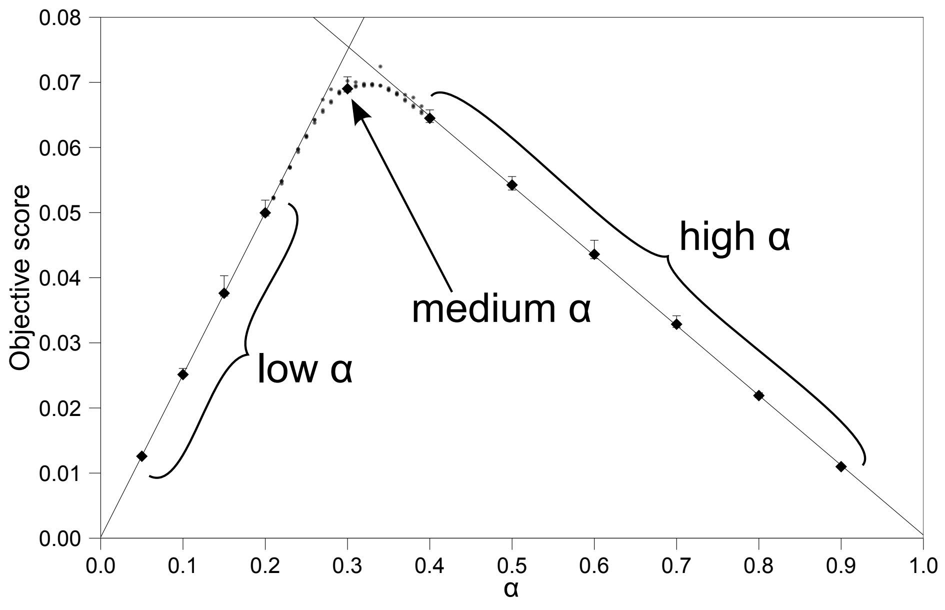


Figure 3.18: The effect of varying α on the optimal objectives. Dark points are the average of 100 optimisations and have max-min error bars (hardly visible). The two lines are linear regression lines through points at low $\alpha = (0.05 \text{ to } 0.2)$ and high $\alpha = (0.4 \text{ to } 0.9)$. The small grey points (appearing similar to a chain or dotted line) are single optimisations giving a higher resolution at medium α values $(0.2 < \alpha < 0.4)$.

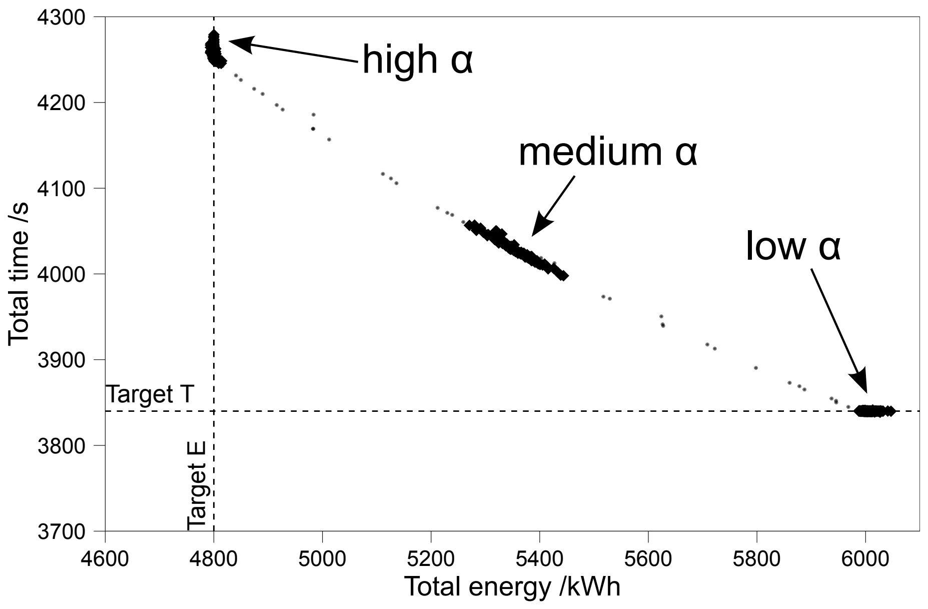


Figure 3.19: Pareto front of total traverse time against total energy consumption. The dark points are shown for consistency with Figure 3.18, and all come from sets of 100 repeats.

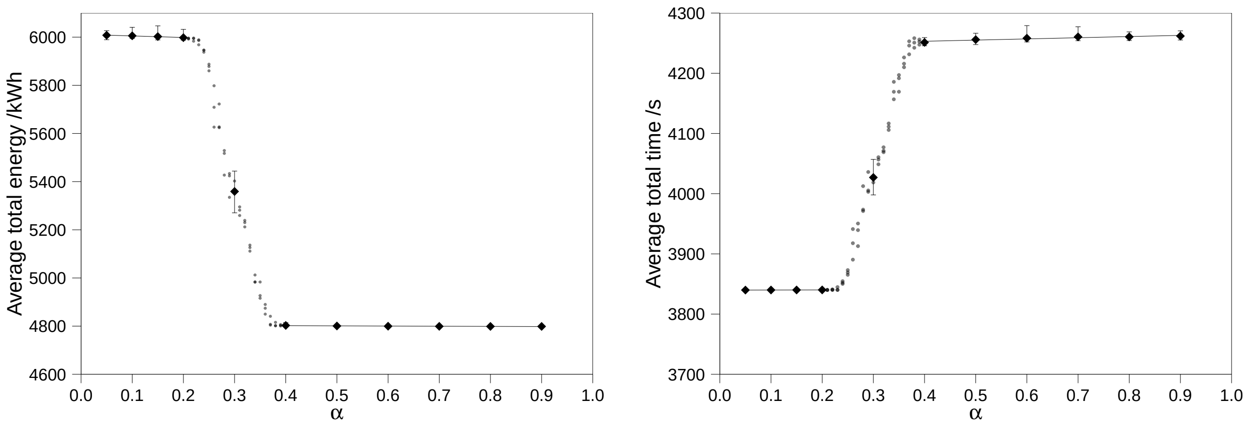


Figure 3.20: Varying the objective weighting, α, causes a step-like response in the optimised solutions found.

misation results [57]. Furthermore, clusters of extreme solutions of min-time and min-energy, as described in Bocharnikov et al. [30], are found for low α and high α, respectively. This suggests that the optimisation is effective, though the small number of intermediate solutions means the components of the objective function respond like step functions with regard to variation in α. Plotting the results in an alternative form this can be seen in Figure 3.20.

The step behaviour requires further investigation under a broader range of conditions, but could be a very useful property in the context of railway operation. While optimising for either shortest travel time, or least energy usage, it would be difficult to timetable trains subject to a continuous range of travel times on a single route. Much easier to manage would be a distinct division into ‘fast’ trains, and ‘energy saver’ trains, with a broad range of optimised driving styles producing one behaviour or the other, i.e. the outcome is resilient to real-world application of the optimised strategy. This concept of resilience of optimised strategies is explored further in Chapter 4.

Table 3.7: The results from applying optimisation G1 and G4 to networks N1 to N5 (100 independent optimisations were carried out for each combination of optimisation and network).

	Objective scores after optimisation							
	None (random initialisation)		G1		G4		Improvement of G4 compared to G1	
Network	Mean	σ	Mean	σ	Mean	σ	Mean (%)	σ ratio (σ_{G1}/σ_{G4})
N1	0.185	0.026	0.091	0.011	0.069	0.0004	23.5	25
N2	0.258	0.032	0.160	0.013	0.1331	0.0005	28.0	28
N3	0.164	0.034	0.065	0.010	0.0457	0.0003	19.4	34
N4	0.244	0.030	0.157	0.010	0.1287	0.0004	32.2	27
N5	0.262	0.025	0.166	0.015	0.1318	0.0006	35.1	26
						Average	27.6	28

3.5.3 Effect of train schedule

A thorough investigation into the scalability of the proposed optimisations is a topic for future research. However, it is important to investigate the characteristics of the optimisations with respect to different train timetables to ensure the results described so far can be generalised and are not just artefacts of the specific timetable defined for network N1. In order to investigate this point, four new networks were defined - each based on network N1 but with changes affecting the scheduling of trains (Figure 3.21). The result of applying optimisations G1 and G4 to each of these networks is given in Table 3.7.

It can be seen from Table 3.7 that even when the optimisations are applied to networks with different timetables, the overall pattern of improvements (first observed in Table 3.6) still hold true - G4 finds better scoring solutions than G1 (by an average of 27.6%) and also does so much more consistently (by an average factor of 28). The smallest improvement in mean optimised score, of 19.4%, is observed for network N3. However, rather than suggesting degraded performance of G4 it is thought this may be caused be a chance

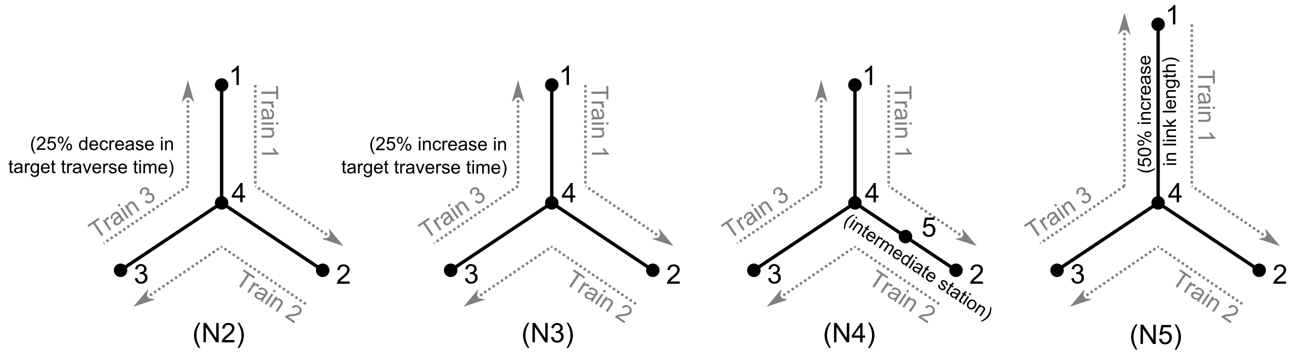

(25% decrease in target traverse time)
Train 1
Train 2
Train 3
1
2
3
4
(N2)
(25% increase in target traverse time)
(N3)
5
(intermediate station)
(N4)
(50% increase in link length)
(N5)

Figure 3.21: Four networks based on network N1 (see Figure 3.1). The associated timetables and energy targets are the same as N1 except for the following changes: (N2) 25% decrease in both target traverse times of train 3, (N3) 25% increase in both target traverse times of train 3, (N4) trains 1 and 2 must dwell at station 5 for at least 30 s and 20 s respectively, (N5) target energy and traverse times are increased by 50% for both the journeys that traverse the longer link between stations 1 and 4.

improvement in the performance of G1 (due to the increased proximity of well-optimised solutions to initialisation - see Figure 3.16). Comparing the relative scores of different networks to N1 we see that the more challenging targets of N2 are consistent with its higher mean score, whereas N3 has more relaxed targets resulting in a lower score. The situation for N4 is slightly more complex, with two obvious factors likely contributing to its increased mean score: the additional stop/starts increases energy consumption and the extra dwells have potential to cause knock-on delays at station 4. Although it is difficult to pick out either as the dominant cause of increased mean score in N4, the energy and traverse time targets for each journey in N5 are equivalent to those in N1 (when normalised by the distance being travelled). Thus, considering all train journeys in isolation we would expect similar optimised scores. However, when optimised considering interactions between trains, the mean score of N5 is significantly higher than that of N1. This suggests that the root cause of the increase in score is from interactions between different trains on the network - in this case the delay of train 3 at station 4 as it waits for train 1 to clear the longer link. The significant effect of interactions between trains when evaluating a timetable highlights the fact that multi-train trajectory optimisation is closely linked to the field of schedule optimisation, particularly if energy consumption is considered, as in Yang et al. [58]

3.6 Chapter conclusions

Several improvements have been proposed and demonstrated to advance the capability of the multi-train trajectory optimisation originally proposed by Yang et al. [44] Two new genetic operators, tailored to the problem formulation, were developed: a less-constraining mutation operation and a procedure to insert and delete pairs of control points. Together, these improvements were shown to optimise an average of 27.6% further than published results when compared with randomly initialised solutions. This was achieved in combination with increased consistency (1/28th of the standard deviation in objective score of solutions), and faster GA convergence (less than one-quarter the number of generations). The resulting optimised trajectories now appear consistent with those expected by optimal control theory.

The improved optimisation consistency allowed a more detailed investigation of the effect of varying α, the weighting between different objectives in the cost function, to be conducted. For the system studied, the components of the objective function respond like step functions with regard to variation in α, causing the optimal objective solutions to switch rapidly between the extreme solutions of minimum time and minimum energy.

Chapter 4

Robust multi-train trajectory planning for real world conditions using a ‘noisy’ genetic algorithm

4.1 Introduction

In Chapter 3 optimisation G4 was proposed and demonstrated to consistently find highly optimised train trajectories for a multi-train system. However, as observed in Section 2.3, the optimum solution is likely to lie on the limit of the feasibility boundary so will be very sensitive to noise [45]. In this context ‘noise’ refers to the many small uncertainties that most current models do not consider but which do exist in reality. This means the highly optimised solutions found by G4 are likely to be difficult to implement in a real system.

There are two possible approaches to address this problem: (i) implement a closed-loop (i.e. real time) optimisation or (ii) find trajectories that are robust to the noise expected in the system being optimised. The first approach has the potential to give better results but requires a fast multi-train trajectory optimisation and is beyond the scope of this PhD. Even for the relatively simple systems studied in Chapter 3 optimisation G4 took around 1 minute to execute so is likely to be too slow to use as a real time optimisation. However, the second approach seems more achievable. This is because G4 uses a genetic algorithm which could potentially be adapted, using the techniques described in Section 2.3.1, to find robust control strategies.

The development of a robust multi-train trajectory optimisation (G5) from G4 is described in Section 4.2. In Section 4.3 G5 is then used to find control strategies optimised for use in systems with different levels of uncertainty in control point application and dwell time at stations. Finally, the performance of closed-loop control is estimated in Section 4.4 and compared to the performance of the robust optimisation.

4.2 A robust multi-train optimisation (G5)

A general methodology for robust optimisation using a GA was presented in Section 2.3.1. This is applied to optimisation G4 to give a robust multi-train trajectory optimisation, hereafter referred to as G5. Two sources of noise are considered: variation in control point (CP) application, and the variation in the dwell time (DT) at stations. These were chosen as prominent examples of type 1 and 2 uncertainty present in many railway systems.

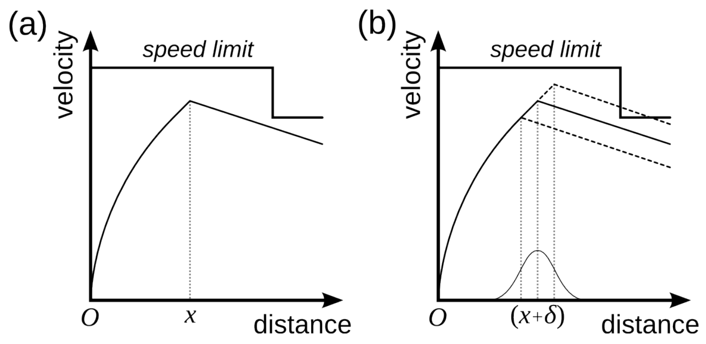


Figure 4.1: Effect of uncertainty in control point application on train velocity trajectories: (a) When applied without any noise, a control strategy may consistently pass close to a feasibility boundary without the possibility of violations arising. (b) When implemented with noise the same control strategy may break the safety constraints, which will result in it being evaluated as ‘invalid’ for that simulation.

4.2.1 Uncertain control point application

A control point (CP) is a location where the train moves between states of traction, speed holding, coasting or braking. Optimisation can be used to establish the best places to make these control changes. However, the planned positions may not be followed accurately, whether by a human driver or automatic control. Adapting the model described in [53] to introduce an example of Type 2 uncertainty into the system was achieved by temporarily adding a zero mean normally distributed random distance to the position of each control point for the duration of each simulation. Instead of the next control action being applied as soon as the train passed the distance specified in the control sequence, the control action was applied with the specified level of uncertainty (Figure 4.1). However, this modification led to two closely related situations becoming possible, both of which required alterations to the algorithms discussed in Chapter 3.

Firstly, each time a candidate control strategy was simulated there was a chance that the trajectories taken did not maintain safe operation (either by exceeding the speed limits or headway constraints). To discriminate against invalid candidates, while maintaining the same population of valid candidate solutions in the modified optimisation, any candidate solution evaluated as invalid was discarded and replaced with a different candidate probabilistically selected from the previous generation. This policy of re-selection (and subsequent re-simulation) is equivalent to using explicit re-sampling to find the expected probability of a candidate being valid, biased towards greater re-sampling of high performing candidate solutions. Instead of the probability of a candidate solution passing to the next generation being proportional to its expected fitness rank, the total probability is now proportional to its expected fitness rank multiplied by the probability that it is valid.

Secondly, it is possible that the 'best' control strategy found by the optimisation in fact relies on very specific deviations on control point application. These deviations would be unlikely to reliably occur in reality which could easily result in invalid operation if the control strategy was implemented without these deviations present. To screen out these cases, the final population was re-simulated without any noise and any candidates found to be invalid were discarded.

4.2.2 Uncertainty in dwell times at stations

Although the running times of trains are themselves subject to some variation, this is within the control of the drivers or dispatchers in most cases. In contrast, station dwell times (DTs) are inherently unpredictable and harder to control [59]. This is largely due to the boarding and alighting of passengers, the speed of which is affected by many uncontrollable variables, such as the number and configuration of train doors, and the configuration of the platform to reduce congestion between alighting and joining passengers. In addition to

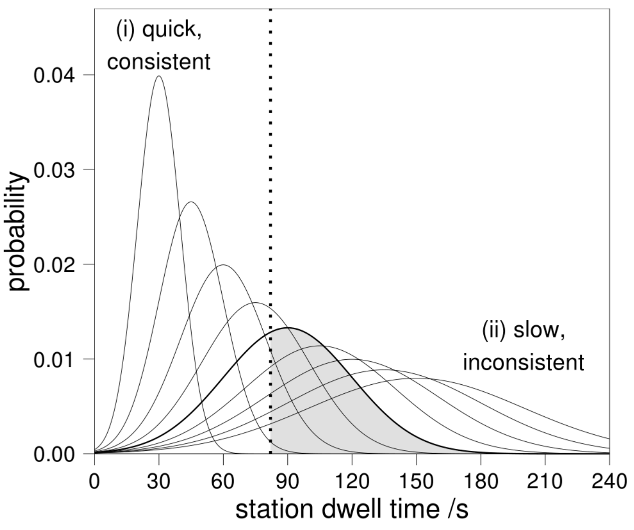


Figure 4.2: Distribution in stochastic dwell times (DTs). The probability distributions used in this system when introducing noise in the DT. The situation dependent minimum DT (dotted vertical line) is dependent on the minimum DT, the scheduled departure time, and train dispatching. Equation (4.1) is used to combine these constraints with a DT probability distribution (in this case, mean = 90 s, sd = 30 s) to give the distribution of possible DTs (shaded area) possible in that specific situation. (i) and (ii) illustrate the two extreme types of DT distribution.

the fleet characteristics there can also be specific local issues such as the familiarity of travellers with the journey, the rate of passenger arrival at the station, and the volume of luggage being carried. For example, an airport station having mostly occasional travellers carrying large items of luggage would be expected to have different DT characteristics to a station with mainly daily commuters without luggage (cf. Figure 4.2(i) and (ii)). Variation in station DTs was introduced into the model as an example of Type 1 uncertainty. Similar to the introduction of CP noise, uncertainty in the DT was implemented by replacing deterministic DTs with stochastic DTs. Modification of the existing algorithms in [53] enabled the optimisation to continue functioning effectively. The departure time of each train was determined using Equation (4.1).

$$\text{departure time} \quad = \quad max\begin{pmatrix}\text{arrival time + stochastic DT,}\\ \text{arrival time + minimum DT,}\\ \text{scheduled departure time,}\\ \text{time of headway conflict resolution}\end{pmatrix} \tag{4.1}$$

Different studies suggest different ways to represent station DTs, including: normally distributed [60], log-normally distributed [59], Weibull distributed [61]. For simplicity this study assumed the stochastic component of the DT was normally distributed. However, once combined with other operational constraints (introduced in Equation (4.1)) the actual distribution of DTs will end up being more positively skewed (see shaded area in Figure 4.2), and so agree better with the majority of the other studies. To enable the uncertainty in DT to be varied smoothly between the two extreme examples shown in Figure 4.2, the standard deviation in the stochastic component of DT was chosen to always be one third of the mean, with distributions having mean =

[30, 45, 60, 75, 90, 105, 120, 135, 150] s and standard deviation = [10, 15, 20, 25, 30, 35, 40, 45, 50] s respectively. This resulted in the series of distributions shown in Figure 4.2, although in real application of the model these could be tuned to the characteristics of a specific station, line, fleet or known passenger behaviour.

Current operational practice is more accurately reflected by defining a schedule, rather than the target traverse times used in G1 to G4. This meant it was necessary to adopt a different cost function, shown in Equation (4.2).

$$\begin{aligned}\text{cost function} = \sum_{i=1}^{i\_max} max(0, \text{arrival time - scheduled arrival})\\ + c_e\Big(\sum_{i=1}^{i\_max} \text{energy consumed on journey}\Big)\end{aligned} \tag{4.2}$$

where c_e is a constant giving the relative value of energy compared to time delays, and i_max is the total number of journeys taken on the network. This is similar to the cost function in [39, equation 1].

If given financially, the value of arriving late is similar to the current UK concept of 'delay minutes', but without any attribution to the causer of the delay. When optimising train movement of the system as a whole, attributing delay is no-longer meaningful as it is the total cost for the system which is being minimised even if this may disadvantage some particular services. This is an implementation issue beyond the scope of the work presented here, but it is expected that improved estimates of optimal system performance could be used to better quantify the costs attributed to different parties in the event of disruption.

One further modification necessary to make the GA procedure function well in a noisy environment was the method of identifying the best candidate solution found. Without noise, the best candidate solution was simply the control strategy that was evaluated as having the lowest cost function score. This ‘all-time best scoring’ control strategy could be found at any time during the optimisation progress. However, in a noisy system the best score found (after a single evaluation of each candidate solution) was very unlikely to identify the candidate with the best expected score. Instead, the best score often turned out to be an outlier from a candidate solution with a fairly mediocre expected score. For example, even when a control strategy causes DT variation to frequently introduce large delays, the distribution of DTs means there is still a small chance that all DTs will be small. A solution evaluated under these improbable conditions will have an artificially high score, predicting high performance which would be unlikely to be realised. To overcome this, the expected score and probability of being valid was estimated explicitly (using Equation (2.4) with $N = 50$) for all the individual candidate solutions in the final population only. These measures were then combined, and the control strategy with the lowest (expected score / expected probability of being valid) chosen as the output of the optimisation. This method of identifying the best control strategy is similar to the effective selection pressure on the population used during the optimisation process. Explicit averaging over the final population required an additional ($N *$ `pop_size`) evaluations to be carried out, but was found to greatly improve the quality of the final candidate solution chosen.

4.2.3 Model parameters

The underlying model in G5 is identical to that used in G4 except for the modifications explained above. The GA parameters used in this investigation are

Table 4.1: Model and GA parameters used in this investigation.

Parameter	value

M (initial size of mutation) /m	start = 200
	end = 0

selection pressure in roulette wheel selection	0.05
`pop_size`	100
number of generations	200
c_e (relative value of energy) /min/kWh	0.0015
N (re-samplings when explicitly averaging final population)	50
minimum dwell time /s	30

shown in Table 4.1. As is often the case with GAs, no systematic optimisation of parameters has been conducted. However, the GA performance achieved is sufficient to demonstrate the validity of the proposed technique for increased robustness. Eiben observes [62] that parameters may have different optimum values throughout the optimisation process, and suggests that a suboptimal choice of parameter function (varying with the number of generations) can often lead to better results than a suboptimal choice of a 'rigid' parameter. Illustrative of this, most GA parameters were kept constant as the optimisation progressed but it was found that a linear decrease in the size of mutation, with each generation, yielded improved performance.

4.3 Investigating performance of the robust optimisation

It is important to draw a distinction between *training noise* (the level of uncertainty during the optimisation process) and *utilisation noise* (the level of uncertainty in the system where the optimised control strategy is applied). Since GAs are not deterministic, each optimisation was repeated 50 times at

Table 4.2: Target schedule for the system modelled (cf. Figure 3.1).

	Scheduled time /s				
	Origin		First stop		Second stop
Train	(arrive)	depart	arrive	depart	arrive
1	(-60)	0	675	735	1482
2	(-60)	0	701	761	1366
3	(-60)	0	599	659	1296

each combination of training noise being investigated. The same set of 50 independently initialised populations was used for the repeats at each noise level to ensure it was the optimisation affecting results and not an artefact of different initial populations. After completion of each optimisation a sensitivity analysis was conducted. The performance of the chosen control strategy was explicitly estimated more accurately, using Equation (2.4) ($N = 500$), at each different level of utilisation noise. Again, at each noise level the same set of 500 utilisation noise instances were used on all 50 repeats to ensure that the optimisation was the only factor changing.

As in Chapter 3, network N1 (Figure 3.1) was used to test the new optimisation. The same number of train routes and stops were retained but, given the use of a new cost function in Equation (4.2), a target schedule was defined based on a scheduled DT of one minute and traverse times with 10% slack (Table 4.2). Provided there are no interactions between trains, flat-out (i.e. maximum permissible speed) control will result in the minimum possible traverse times. An estimate of flat-out control was made by conducting 100 optimisations to a target schedule with: unachievably short traverse times (100 s), no energy cost ($c_e = 0$), and long DTs to avoid interactions between trains (1000 s). Ten percent was then added to the minimum traverse time found for each journey to create the schedule in Table 4.2.

Although only nominal estimates of CP and DT noise levels and the relative value of energy and time (c_e) were used, the overall pattern of results observed is expected to remain the same if different, more system specific, values were used instead.

4.3.1 No training noise

The first series of optimisations was carried out without any training noise during the optimisation process. The results of this are shown in Figure 4.3. This is equivalent to a conventional optimisation developed for application in ideal conditions (i.e. when there will be no uncertainties during utilisation). With that in mind, it would be expected that its performance when utilised with no control point or DT noise would be good, but would deteriorate markedly under real conditions. For the system studied, the utilisation DT distribution was found to have no effect on the validity of control strategies, and the variation in utilisation CP noise had negligible effect on the expected score of a control strategy. These relations did not appear to change for different combinations of training noise so are not discussed further.

From Figure 4.3(a) it can be seen that the probability of an optimised control strategy being valid drops quickly as CP application noise is increased, and stays at a low level. The optimised solutions found by the GA are very close to the constraints imposed to ensure safe operation (in this case velocity limits). When utilised in a system with no noise they always keep the safety constraints and are therefore considered valid solutions. However, as soon as a small amount of CP noise is introduced during the utilisation of the control strategies the probability of speed limit violations occurring (through a situation similar to the one illustrated in Figure 4.1) becomes high. Far from being surprising, this lack of robustness is exactly the behaviour we

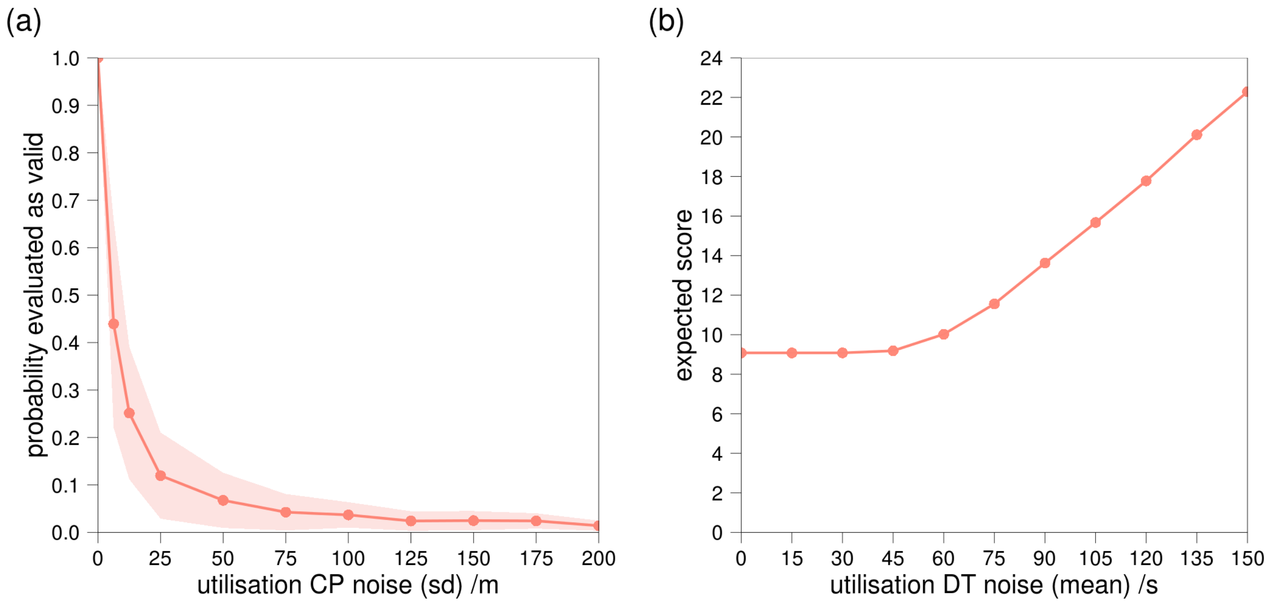


Figure 4.3: The average robustness and expected score of control strategies found under conditions of zero training noise were investigated by simulating their utilisation at different noise levels. Shaded areas show one standard deviation. (a) Effect of variation in utilisation CP noise on the probability the control strategy is evaluated as valid (utilisation DT noise = 0 s). (b) Effect of variation in utilisation DT noise on the expected score of operations (utilisation CP noise = 0 m).

would expect from near optimal solutions to the noiseless problem [45]. In other words, by not considering uncertainty (i.e. training CP noise) during the optimisation we have unwittingly sought non-robust solutions (with respect to CP noise).

Similarly, Figure 4.3(b) shows that, above a certain threshold, increased utilisation DT has an almost linear effect on the expected score of a control strategy, i.e. increasing poor performance. In seeking to minimise energy consumption the optimisation has found solutions that make full use of the scheduled traverse time on each journey (since losses due to air resistance are reduced at lower speeds). This means that recovery time has been minimised so any late departure will cause a late arrival (with this effect amplified as delays propagate across the network). Below DT noise = 45 s this lack of recovery time is not an issue because the stochastic DT has a very low probability of being greater than the scheduled DT (60 s) - see Figure 4.2. However, above DT noise = 60 s the majority of dwells are extended and, since there is minimal recovery time, any increase in DT increases the expected score of evaluations. Again, by not considering any uncertainty during the optimisation we have unwittingly sought solutions that are by their very nature non-robust.

4.3.2 Control point application training noise

The second series of optimisations was carried out with different levels of training CP noise during the optimisation process, the results of which are illustrated in Figure 4.4.

Figure 4.4(a) shows that increasing the CP training noise leads to the optimisation finding control strategies that are substantially more robust to variation in CP application. However, from Figure 4.4(b) it can be seen that when utilised at a zero CP and DT noise there is a small increase in cost for

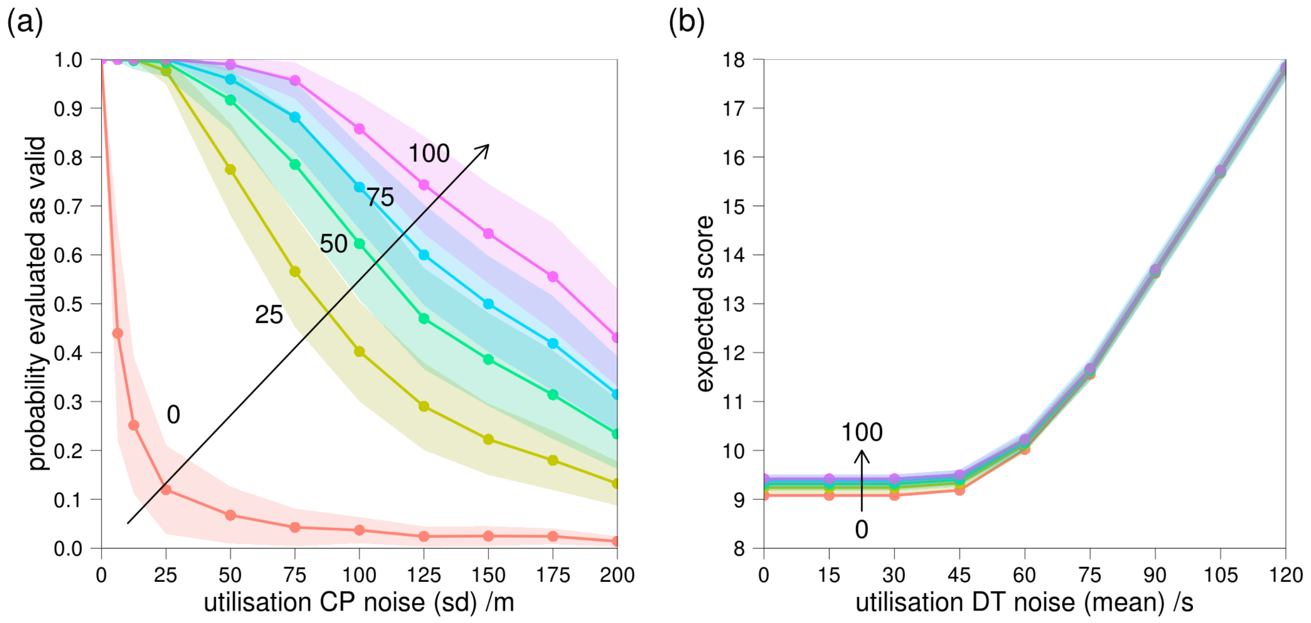


Figure 4.4: Control strategies were optimised using different levels of CP training noise (labelled on the figure), but without any training DT noise, during the optimisation process. The average robustness and expected score of these control strategies were investigated by simulating their utilisation at different noise levels. Shaded areas show one standard deviation. (a) Effect of variation in utilisation CP noise on the probability the control strategy is evaluated as valid (utilisation DT noise = 0 s). (b) Effect of variation in utilisation DT noise on the expected score of operations (utilisation CP noise = 0 m).

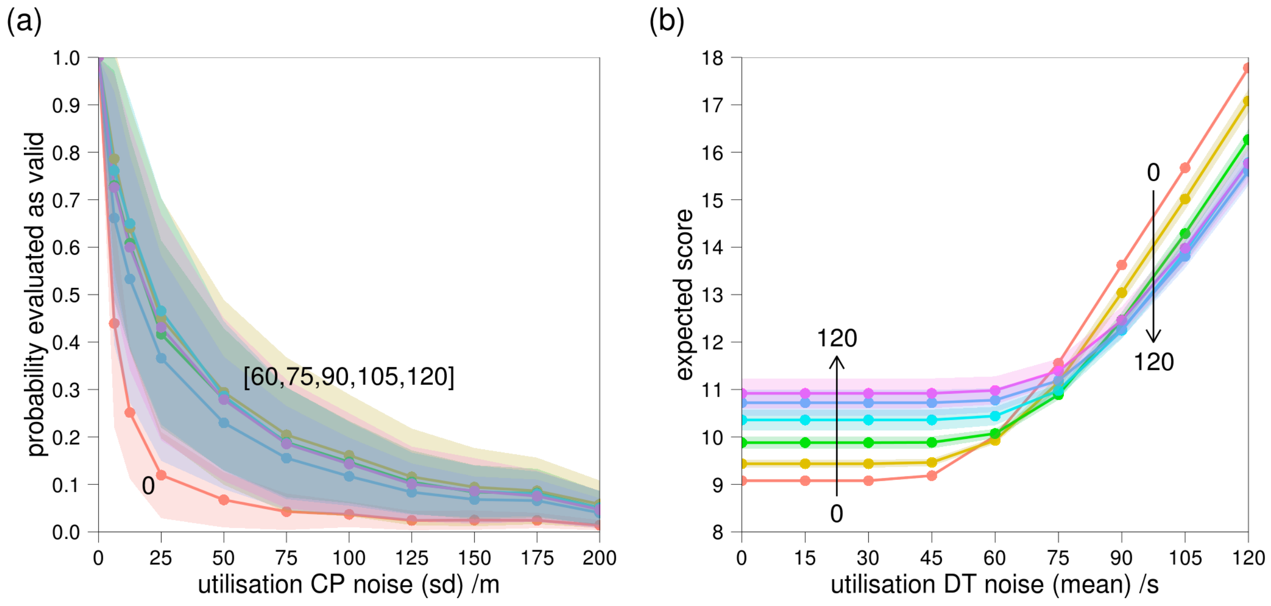


Figure 4.5: Control strategies were optimised using different levels of training DT noise (labelled on the figure), but without any control point (CP) training noise, during the optimisation process. The average robustness and expected score of these control strategies were investigated by simulating their utilisation at different noise levels. Shaded areas show one standard deviation. (a) Effect of variation in utilisation CP noise on the probability the control strategy is evaluated as valid (utilisation DT noise = 0 s). (b) Effect of variation in utilisation DT noise on the expected score of operations (utilisation CP noise = 0 m).

this increase in robustness. Even in the worst case of this, seen at a utilisation CP noise of 100 m, increasing the CP training noise from 0 to 100 m causes the probability of the control strategy being evaluated as valid to increase from 0.04 to 0.86 (>2000%), but the expected score to increase by only 2.3%.

4.3.3 Station dwell time training noise

The next series of optimisations were carried out with different levels of DT training noise during the optimisation process. The results for training DT noise (mean) = [30, 45] and [135, 150] s were found to be almost identical

Table 4.3: The effect of utilisation noise on the expected benefit from different levels of training noise (variation in DT noise only, CP noise = 0 in all cases).

	Expected % score reduction (relative to training DT noise = 0)	
Training DT noise (mean) /s	utilisation DT noise = 0	utilisation DT noise = training DT noise
45	-1	0
60	-4	1
75	-9	6
90	-14	10
105	-18	12
120	-20	11

to training DT noise (mean) = 0 and 120 s respectively so are omitted from Figure 4.5. For the first case, this is because the training noise level is too low to have a noticeable effect - the vast majority of random DT instances are less than the scheduled DT used in the system (60 s) and therefore rarely affect the actual departure time (see Equation (4.1)). In the second case, this is because the training noise level is too high - the optimisation can no longer distinguish genuine improvements in control above the noise. This is discussed in more detail in Section 4.4.

It can be seen from Figure 4.5(b) that increasing the DT training noise leads to solutions that have a lower expected score when utilised at high DT noise (i.e. are more robust). Figure 4.5(a) shows a secondary benefit to the control of slightly increased robustness to variation in CP accuracy. In this case, introducing one type of noise has led to the system becoming more robust to another type of noise that was not selected for during optimisation. However, the increase in robustness is accompanied by an increase in expected score when utilised at low DT noise - seen in Figure 4.5(b). This suggests that the optimisation is working effectively, because the solutions being found are

Table 4.4: Average properties of the control strategies resulting from different combinations of training noise (all utilised at, CP noise = DT noise = 0).

	Training noise (CP_DT)			
	0_0	100_0	0_90	100_90
Mean speed /ms^{-1}	45.6	45.7	47.1	46.6
Mean journey time /s	658	656	636	644
Mean journey energy /kWh	1008	1038	1148	1152

well suited to the environment they were trained for, but that particular care should be taken to make sure that the training noise matches the noise level at which the solution with be utilised. This is highlighted in Table 4.3 where the performance when the utilisation noise matches the training noise is found to be much better than when the utilisation noise is fixed but the training noise varied. In situations where the utilisation noise levels of a real system are not known, estimates must *always* be made when implementing the training noise. It follows that all non-robust optimisations make the (usually implicit) assumption that noise levels on all parameters are zero. For many situations, particularly metro applications, this may be an acceptable approximation but it is unlikely to hold in complex, interconnected, stochastic systems such as a busy mainline rail network.

The reason for this trade-off, between the score of solutions (utilised DT noise = 0) and their increased robustness, can be understood by looking at the traverse times of the trains and the corresponding energy consumptions. For convenience the convention CP_DT will be used to describe the training noise levels used during optimisation (e.g. 0_90 denotes a training CP noise of 0 metres and a training DT noise of 90 seconds). The average scheduled journey time in the system modelled is about 661 seconds and it can be seen from Table 4.4 that the average journey time with 0_0 gives about 3 seconds of recovery time per journey. In-fact, on closer inspection of the schedule in Table 4.2 we observe that the first journey by train 1 is scheduled to arrive after train 3 has begun its second journey. However, we can also see from Figure 3.1 that these journeys take place on the same section of track (joining stations 1 and 4). So, to maintain the headway constraints (i.e. avoid a head-on collision) train 1 must arrive before train 3 departs. The optimisation consistently achieves this by controlling train 3 such that it arrives 16 seconds early. Therefore, the average journey time with 0_0 effectively has less than 1 second recovery time. This results in 0_0 keeping the schedule while having the lowest average speed and therefore the lowest energy consumption in Table 4.4. Previously this would have been considered a success, since the service is punctual and energy efficient, but when considering robustness we find that it is actually a 'brittle' solution, with utilisation noise rapidly leading to sub-optimal performance. It is also a good reminder that trajectory planning and timetable/schedule optimisation are closely coupled problems [9].

If the training DT noise is increased to 90 s (training noise 0_90) then the average recovery time increases to about 23 s per journey. This allows punctual operation to be maintained in systems where there is a significant probability that DT will be longer than scheduled, but at the cost of running faster and using slightly more energy. Interestingly, fast running (in order to build up recovery time) is similar to typical driver behaviour [63] but in this case has been found by a direct optimisation, which has no 'understanding' of the system it is optimising or prior knowledge of existing operational concepts.

The recovery times used in the 0_90 solutions are different for each journey and are not trivial to find. They cannot be estimated simply by considering the scheduled traverse times. Instead, they depend on the different speed limits and headway restrictions experienced by each train as it moves through the network. In particular, headway restrictions will determine how delays propagate across the network causing some train journeys to be more susceptible to knock-on delays than others.

4.3.4 Both control point application and station dwell time training noise

So far the proposed procedure has shown success in finding solutions with increased robustness to a single type of noise - CP application and DT, respectively. However, in real systems there may be uncertainty in many parameters simultaneously, so it is important to investigate the performance of the proposed procedure in this situation. The performance when high levels of CP and DT training noise are used simultaneously is shown in Figure 4.6.

It can be seen from Figure 4.6(a) that the robustness of control strategies to noise in CP application is predominantly influenced by the training noise level of CP used during optimisation. In terms of maximising robustness to CP variation the performance of the 100_90 optimisation is very close to the performance of the 100_0 optimisation, and is actually marginally better. In contrast, Figure 4.6(b) shows the performance of the 100_90 optimisation is similar, but slightly worse, than the 0_90 optimisation. Including both training noises simultaneously has led to worse performance than just applying training DT noise on its own. It is thought this is due to degradation in the optimisation performance as the total noise level increases. This idea is discussed in more detail in Section 4.4. However, the performance of the 100_90 optimisation is still an improvement over the 0_0 optimisation. This shows that the proposed optimisation procedure is capable of finding solutions that are more robust to two different types of uncertainty simultaneously.

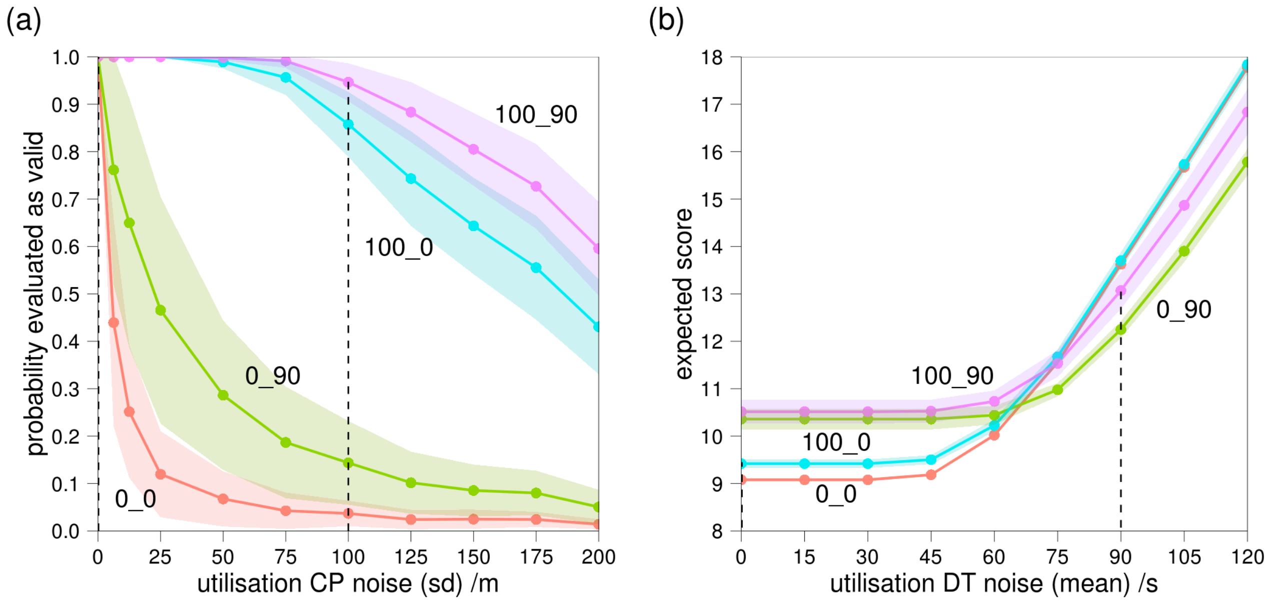


Figure 4.6: The performance of optimised solutions found with different combinations of training noise (CP_DT): no training noise (0_0), higher levels of CP and DT training noise applied separately (100_0 and 0_90 respectively), and also simultaneously (100_90). The dashed vertical lines emphasise the level of the training noise used.

4.4 Comparison with closed-loop performance

The proposed optimisation procedure for robust trajectory planning (G5) gives clear benefits over its non-robust counterpart (G4). However, since they are concerned with trajectory planning, both G4 and G5 are open-loop optimisations. This means that when being utilised, the control of each train is assumed to be independent of the actual conditions being experienced - the optimisation is trying to find one solution that works well over all probable situations. For example, if the control strategy is blindly followed, a very late train will not increase its speed to make up time and so will incur a larger delay penalty than necessary. Conversely, a train which finds itself running ahead of time will not slow down and may therefore consume more energy than it may otherwise have done. In a noisy system a good closed-loop optimisation has the potential to perform better than any open loop optimisation but must be carried out in real time. Closed-loop control would demand a communications infrastructure and real time optimisation which is not yet available on most mainline railway networks. A good closed-loop optimisation has the potential to perform better than any single open-loop solution, but by how much?

When utilising a robust solution, the expected performance was estimated using the Monte-Carlo method, Equation (2.4). It is possible to estimate the lower bound in performance of closed-loop optimisation by applying the open-loop optimisation separately to each noise level instance. This was carried out at each level of DT noise while CP noise was kept at zero. The same 500 utilisation instances (used when explicitly averaging closed-loop solutions) were used at each noise level. No optimisation repeats were carried out due to the increased computational cost of closed-loop estimation; an entire optimisation must be carried out for every single sample taken during the Monte-Carlo

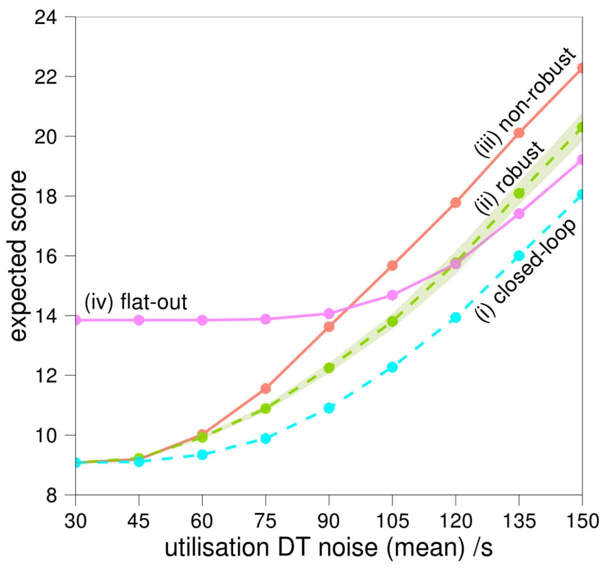


Figure 4.7: The effect of utilisation DT noise on the expected score of solutions found using different types of optimisation: (i) closed-loop estimates, (ii) robust open-loop (training noise = utilisation noise), (iii) non-robust open-loop (training noise = 0), (iv) flat-out operation. Both (ii) and (iii) are the average of 50 optimisation repeats, while the other types of optimisation were not repeated.

evaluation. It is important to emphasise that this method estimates the lower bound of closed-loop performance and the optimisation actually has information on all the DTs that will happen in the network. In reality a closed-loop optimisation would only have definite information on DTs that have already happened.

For the system and noise levels studied, Figure 4.7 presents the estimated performance of a closed-loop optimisation. As would be expected, the closed-loop optimisation outperformed both the non-robust and robust open-loop optimisations over all DT noise levels. However, for this system, the performance of the robust optimisation appears to be about half way between the performance of the non-robust and closed-loop optimisations. In fact, between

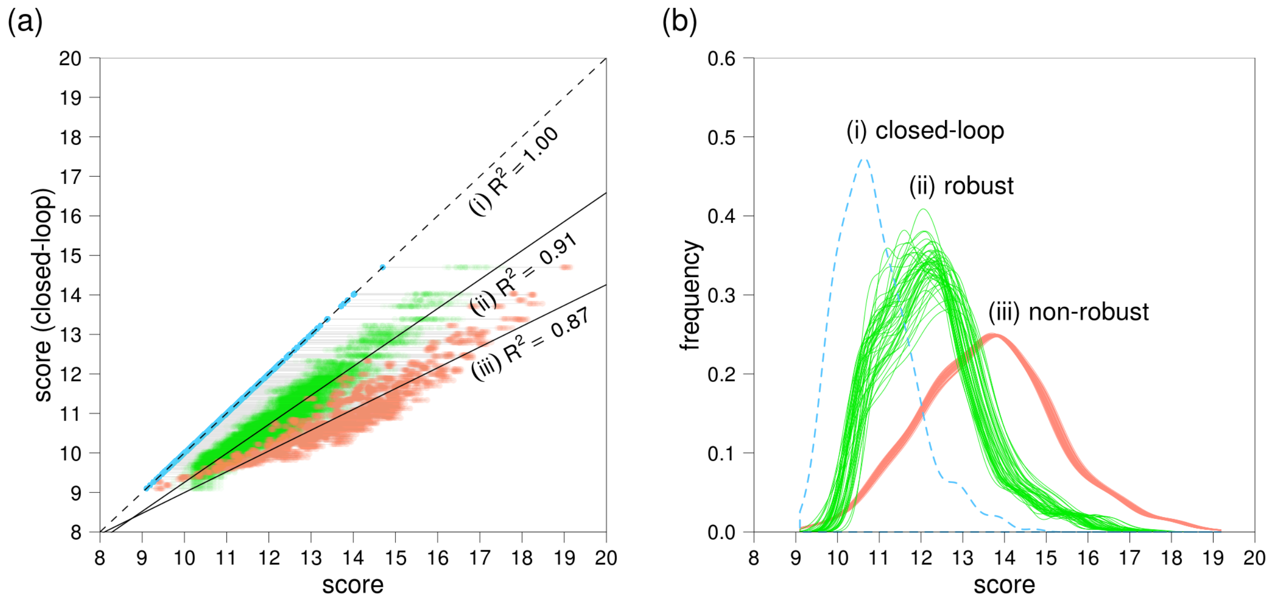


Figure 4.8: The performance of different types of optimisation when evaluated with the same set of utilisation noise instances ($N = 500$; CP noise = 0 m, DT noise (mean) = 90 s): (i) closed-loop estimates, (ii) robust open-loop (training noise = utilisation noise), 50 optimisation repeats, (iii) non-robust open-loop (training noise = 0), 50 optimisation repeats. (a) Scores at different instances of the utilisation noise against the closed-loop score for that utilisation instance. Horizontal grey lines link the scores from each instance of utilisation noise. (b) The distribution of scores from each type of optimisation. Robust open-loop solutions are about half way between the closed-loop estimate and the non-robust open-loop solutions.

utilisation DT noise (mean) = 60 to 150 s, the robust optimisation was able to find open-loop solutions giving an average of 44% of the benefit afforded by closed-loop control (relative to the non-robust open-loop solutions). This relative benefit peaked at 55%, at a utilisation DT noise (mean) of 105 s.

In the system studied, energy is inexpensive relative to the cost of delays (2000 kWh:3 minutes delay) so at high levels of DT noise we expect the optimal open-loop solution to approach that of flat-out operation as this will minimise delays. The performance of this solution is given by Figure 4.7(iv). At low utilisation DT noise the flat-out solution uses a lot of unnecessary energy so is much worse than the solutions found by the robust optimisation. However, at high utilisation DT noise the flat-out solution actually performs better than the robust optimisation as it results in fewer late arrivals.

This illustrates an important limitation of the proposed optimisation procedure: when noise levels are too large (relative to the variability in the cost function) the optimisation will struggle to converge on the robust optimum. This is a known effect [64] and is likely to result in deterioration of both the speed of convergence and quality of the final solution found. Re-sampling, both explicitly and implicitly (through increasing the population size), should improve convergence of the GA but eventually the limit of computational resources will be reached. Although not ideal, it is thought this lack of convergence may be advantageous when seeking an open-loop solution for application in a real system. Any non-robust optimisation, even when equipped with the most accurate deterministic cost function evaluation, may appear to find a good solution but if this is then implemented in a noisy system it is likely to result in poor performance. In contrast, the proposed robust optimisation (equipped with accurate estimates for all uncertainty distributions present in the system) should either find a practically useful solution or will not converge.

If the optimisation does not converge for a particular system then this is likely to be because the uncertainties being modelled are too large. If uncertainties cannot be reduced then lack of optimisation convergence is likely to indicate that that system is not well suited to open-loop control or else the optimisation horizon is too large - something non-robust control will never indicate.

Since each point in Figure 4.7 shows the aggregated optimisation performances (i.e. the mean and standard deviation in the expected score over N repeats) it is instructive to consider these results in more detail. The disaggregated data behind three points of Figure 4.7, from lines (i) to (iii) at utilisation DT = 90 s, is presented in Figure 4.8. At this noise level the same 500 instances of utilisation DT noise were used for the Monte-Carlo method when evaluating the performance of each solution found. Within each of these utilisation instances, marked by the horizontal grey lines on Figure 4.8(a), it becomes apparent that the open-loop solutions found by the robust optimisation are consistently better than those found by the non-robust optimisation. The only exception to this seems to be for a small number of instances, at very low scores, where all DTs happen to be almost entirely unperturbed. Figure 4.8(b) shows the distribution of scores resulting from each of the 50 solutions found by both the robust and non-robust optimisations. A distribution is also shown estimating closed-loop performance but it should be emphasised that this does not represent a single solution. The tight overlap of distributions shows the non-robust optimisation to consistently find very similar solutions, whereas the robust optimisation has a larger variation in the shape of distributions. However, solutions found by the robust optimisation have distributions skewed towards lower (better) scores.

4.5 Evaluation of G5

In seeking to increase energy efficiency, driver training [65] and Driver Advice Systems (DAS) [63] often use the concept of a 'golden run' - a perfectly controlled train journey that yields optimum energy efficiency. The 'golden run' has essentially represented the non-robust, open-loop, single-train optimal solution for a train journey. This has obvious limitations but the one big advantage is that open-loop solutions can be implemented without the communication infrastructure and real-time optimisation required for closed-loop control. Despite not being as good as the closed-loop performance, the robust optimisation at DT noise (mean) = 90 s still has an expected score 10% lower than its non-robust counterpart. Since the score is intended to be proportional to the operational cost of the robust solution it may still be worthwhile implementing, conveniently replacing the 'golden runs' currently used in driver training and DAS.

4.6 Chapter conclusion

When planning train trajectories it is important to consider the robustness of control strategies if they are to be implemented in real systems. Real systems contain many uncertainties, such as the accuracy of control point application or variations in station dwell times. If these are not considered during the optimisation process then it is unlikely that the control strategies found will be robust enough to perform as predicted in real operation. Similarly non-systematic approaches (e.g. excessive recovery time built into schedules, or driving trains aggressively in an effort to keep to the timetable) may be over conservative and reduce capacity or increase energy use unnecessarily. A new, genetic algorithm based, optimisation procedure has been described which seeks to find robust solutions to the multi-train trajectory planning problem. The procedure is easily generalisable to include many different uncertainties in the system. Here it was shown to be effective in finding robust control strategies in the presence of two different types of uncertainty: the accuracy of control point application, and variation in station dwell times. These uncertainties were first considered separately, before it was shown that they could be considered simultaneously in the optimisation and still achieve similar levels of robustness. For both types of uncertainty investigated, a trade-off between the robustness and the expected score of the solution was observed, reminding us that robustness is not cost free. This means that for best results the training noise level used during the optimisation progress should reflect, as accurately as possible, the noise level that will be experienced when the optimised control strategy is utilised. The performance of a closed-loop optimisation was also estimated and, as would be expected, this achieved better performance than open-loop solutions found by both the non-robust or robust optimisations. However, for the system and noise levels investigated, the robust open-loop solutions were found to afford up to 55% of the benefit of closed-loop control (compared to non-robust solutions). This suggests the proposed robust optimisation may be worth further investigation, especially considering that open-loop solutions can influence implementation (e.g. via DAS) without the communication infrastructure and real-time optimisation required for optimised closed-loop control.

Chapter 5

A massively parallel multi-train simulator for accelerating population based heuristic optimisations

5.1 The need for a new model

The usefulness of the model first proposed by Yang has been shown in the previous chapters. However, as this work was undertaken, some limitations also became apparent. Many of these limitations are surmountable by modification of the existing model but, as is detailed below, others are more fundamental and require extensive change.

5.1.1 Advantages of the model used in G1 to G5

One nice property of Yang's formulation is that the trajectory each train follows is exclusively determined by its own control strategy. This causes restrictions on the train trajectory to change depending on the position of other trains. How each train reacts to this information (i.e. an autonomous agent behaviour) further influences the final result obtained. The effect of this is that headway constraints are enforced by modifying the trajectories of trains whilst leaving their control strategies unchanged. In contrast, the formulation used in G1 to G5 only allows control strategies that obey the headway constraints. This enables the optimisation to search for the best trajectory to achieve a safe operation. For example, there are many ways to stop at a red signal so the optimisation can seek to find the most efficient one or may be able to avoid approaching the red aspect altogether.

A 'journey' can be defined as the movement of a train between two stops. On Yang's graph-based network model this means each edge of the graph equates to a journey, each of which is simulated sequentially in the order they were scheduled. This means that the partial separation of signalling and simulation also enables whole train journeys to be simulated without interruption maximising the probability of cache hits and therefore fast computation (this concept is explained more in Section 5.2.1).

Also, since Yang's formulation is a time-stepping model (assuming linear acceleration over a small time interval) it is trivial to model non-linear traction, braking, resistance and efficiency characteristics. This is much more difficult using explicit methods [10].

5.1.2 Limitations of model used in G1 to G5

One of the advantages of Yang's formulation is the fact that a train's control strategy directly determines its trajectory. However, this is implemented using hard constraints which leads to several major disadvantages.

Firstly, there is a potential of many journey simulations to be wasted. If a journey is simulated but found to violate the constraints, it must be re-simulated until a valid control sequence is found. This means that computational resources are being wasted as many calculations may be carried out only for the results to be discarded. In order to minimise this Yang initially suggested using very restrictive mutation operations which guarantee not to invalidate the control of later journeys when changing earlier ones. This mutation operation was analysed in Chapter 3 and found to adversely affect the performance of the optimisation.

Secondly, random initialisation may be very time consuming. For an arbitrary network, schedule and interlocking there is not even a guarantee that a valid control strategy exists. Yang's initialisation algorithm appears to work well for the systems studied as it quickly found valid solutions. However, this random initialisation was found to take a significant time as the optimisation became more complex. For example, using a larger number of control points during initialisation increased the number of attempts before the population was filled with valid solutions. Each failed attempt was discarded, again wasting calculations.

Thirdly, it was not able to model junctions. An indirect implication of simulating whole journeys is that headway constraints cannot be checked through nodes. This is because the journeys on the other side of the node may not have been simulated yet, and means that each train must come to a stop at every node on its route. This prevents nodes from accurately representing junctions, as trains are not be able to pass through them at speed. Since, the interaction of trains at junctions can often be the ‘bottleneck’ in railway networks [66] it is important to be able to represent them in a multi-train trajectory optimisation.

Finally, the order of departure from each node is pre-determined. This means the optimisation relies heavily on the input schedule being optimised. Since trajectory optimisation and scheduling are really two parts of one larger optimisation it would be desirable if they could be considered simultaneously [9].

There are also a number of other limitations of the model initially proposed by Yang. These are less fundamental than the limitations discussed so far, so could have been added to the existing formulation. However, since a new model will be developed it is helpful to consider other areas where it could improve upon the model used in G4 and G5.

Firstly, although more detailed than many other models [30, 33], the train control available in the current formulation is relatively limited. It is restricted to the position of switching points between traction-coasting pairs, which limits the areas of the search space that can be described. For example, braking is only possible when coming to a stop at the end of a journey. So if a drop in line speed requires a train to slow, then the only mechanism available to achieve this is coasting. This leads to train trajectories that are slower than if braking was also possible. Although such trajectories are inherently energy efficient, journey times, punctuality and capacity are higher priorities on most modern railways networks. Even if they weren’t it is preferable for the optimisation to have the potential to find a solution over the whole search space. Analytical results have shown five operational modes are sufficient to describe the optimal control of a single train: maximum traction, tractive speed holding, braking speed holding, coasting and maximum braking. However, despite the many simplifications made (see Section 2.1.1), the author has not found any similar analytical result for a multi-train system in open literature. Such a result would depend on the signalling system used as well as the (potentially different) dynamics of all interacting trains. As such, it may be that optimal control requires intermediate levels of traction and braking, which are not currently included in the model.

Secondly, algorithms for a number of processes were not fully described by Yang. Notably, the methods for establishing 'the proper site for using braking operation' and checking speed limits were not explicitly given. In the initial implementation of the model both of these consumed a relatively large percentage of the total computation time. A naive implementation of speed checking in G1 took over 30% of the total CPU time. It involved making a linear search through the array of speed limits for the current link after every time-step was taken. When this was identified as a problem it was replaced with a simple but more efficient algorithm that took under 1% of the total CPU time. In this improved implementation the current speed limit and position of the next change in speed were both recorded after each linear search. This meant that the next linear search of speed limits was only triggered when the specified position had been reached. While this example is a relatively minor thing in itself, it highlights the fact that whenever a process is carried out a large number of times (e.g. every time-step of the simulation) its implementation must be carefully considered. Related to this, track geometry such as gradients, curves, or tunnels was not considered in the original formulation. Since most rail networks are not completely straight and flat these elements must be considered if realistic systems are to be modelled. It would be desirable for a general methodology to be proposed that could efficiently include any property of the track that varies with distance.

Finally, no parallelisation strategy was proposed in the initial formulation. For most heuristic optimisations solution evaluation is the most time-consuming step so decreasing the time taken can bring many benefits. Most obviously, if the number of iterations are kept constant then the optimisation can be carried out faster. This can allow the optimisation to be applied in different situations (e.g. a slow optimisation may only be suitable for planning purposes whereas the same solution found faster might open up the possibility of real time control). Alternatively, the same amount of time could be spent on the optimisation and more iterations could be carried out. This usually increases the quality of optimisation by finding more highly optimised solutions. Similarly, the time and number of iterations could be kept constant but the size of the optimisation population could be increased. This helps optimisations avoid local minima and more reliably find good solutions - particularly important in more complex problems.

5.1.3 Conclusion

As discussed in 5.1.1 and 5.1.2, a next generation simulator should:

- uses a train's control strategy to directly determine its trajectory
- can easily incorporate non-linear traction, braking and efficiency characteristics of trains
- makes better use of computational resources by not discarding information stored in invalid solutions
- can perform efficient random initialisation on more complex systems

- can simulate trains moving through junctions (allowing it to model more realistic networks)
- has the option to integrate trajectory optimisation and timetabling
- uses a greater range of control actions allowing it to explore more of the search space
- can efficiently include any property of the track that varies with distance (e.g. gradients, curves, and tunnels)
- is suitable for parallel computation

At the time of writing the author is not aware of any multi-train simulators that meet that above criteria and are either open source or well documented in literature. Therefore, before progressing further with optimisation algorithm development, the decision was made to develop a multi-train simulator suitable for accelerating population-based heuristic optimisations.

5.2 Introduction to GPUs and CUDA

Once the logic of a computer program is fixed small changes in run time can be made through compiler choices and code optimisation. However, in broad terms, in order to solve the same problem faster it must be run on more powerful hardware. The Central Processing Unit (CPU) of a computer is conventionally where these calculations take place. However, recently Graphical Processing Units (GPUs) have become easier to use for general purpose computing. In a desktop environment they offer increased computation power over CPUs at an affordable price. However, most existing algorithms discussed in the field of trajectory optimisation have be sequential so are not well suited to implementation on a GPU. To help understand why this is the case (and ultimately how to devise efficient parallel algorithms) a brief introduction to GPU architecture is given. Since many of the same optimisations used by CPUs are employed to some degree by GPUs, this section starts with an overview of CPU design principles in 5.2.1. This is followed by a comparison with GPU hardware design in 5.2.2, which also introduces the CUDA programming model. Finally, principles for efficient GPU algorithms are discussed in 5.2.3. Readers already familiar with these concepts may wish to go straight to Section 5.3.

5.2.1 Background on CPU architecture

Latency is the time elapsed between an instruction being issued and its execution being completed. Sections of an algorithm that are executed in serial are referred to as threads, and CPUs are generally optimised to provide *low latency on a single thread.* In general each CPU core has a Single Instruction stream and a Single Data stream (SISD) so low latency on a single thread is achieved by maximising the performance in three main areas[67]:

- Clock speeds
- Execution optimisation
- Caching

Clock speeds determine the rate at which each instruction can be executed. The processor 'clock' produces a square wave which ensures that each execution has finished before the next one begins. The higher the frequency of square wave a processor can operate at, the less time each instruction takes to execute. However, increasing the clock speed also increase the heating of components. This heat must be dissipated so, although higher clock rates are possible with special cooling equipment[68], the clock speed of mainstream air cooled processors has levelled off at around 4-5 GHz [69].

Execution optimisation seeks to maximise the utilisation of the processor hardware. There are several stages to executing a single command so if a naive approach was taken, and each instruction executed sequentially, then most of the processor would be idle most of the time. As a simple example we can consider the case where each command must first be fetched, then decoded, than executed. If each of these stages takes one clock cycle to complete then a processor could execute one instruction every three clock cycles. To avoid this problem most modern processors use an instruction pipeline, where execution of the next instruction can begin before the previous instruction has finished. In the above example this would allow up to 3 instructions to be in the pipeline at the same time, potentially tripling the throughput of the CPU. However, this brings its own challenges as later instructions may depend on the results of earlier ones or conditional branching (e.g. 'if statements') may change which instructions are executed. There are many complex techniques to address this, such as out of order execution and branch prediction, but they all seek to minimise the time processor components are not doing useful calculations.

Caching reduces the average time it takes to read data from memory. Accessing Random Access Memory (RAM) is relatively slow compared to the processor, so it may take tens of CPU clock cycles for data requested from RAM to become accessible to the CPU [70, chap. 5]. If not addressed this would usually be the limiting factor to execution speed and undo much of the gain brought about by having high clock speeds and good execution optimisations. Faster types of memory are available but they are expensive so can't be used in large amounts. When one piece of data is requested from RAM most modern architectures will actually copy a larger block of data to the cache - a faster type of memory within the CPU. Although this does not speed up the first memory request, consecutive instructions often access data stored close together in memory. If the next request is for data already stored in the cache then it can be serviced much faster. On average this leads to a reduction in the time the processor spends waiting while data is fetched and therefore decreases the overall latency on the CPU.

The techniques discussed above give insight into how CPUs have become so fast at implementing serial algorithms. However, using current technology there is a practical limit to the speed a single core can process instructions. About a decade ago improvements in both CPU clock frequencies and execution optimisations began to stagnate and since then the trend has been towards CPUs with multiple cores and parallel execution. [67] This means significantly accelerating the simulation of control strategies would require efficient parallel algorithms to be developed even if CPUs remained the target architecture.

5.2.2 GPU architecture and the CUDA abstraction

In contrast to CPUs, Graphics Processing Units (GPUs) started as hardware dedicated to accelerating the rendering of computer graphics and have always been highly parallel. The desire for real time graphics (driven by the video games industry) requires the ability to rapidly update all pixels on a screen. This can be a very computationally intensive task but is made up of many small independent calculations. To address this, GPUs have been designed to maximise the *aggregate throughput over many threads*, rather than minimise the latency on a single thread. Over time more non-graphics functionality has been added to GPUs until they have become programmable parallel processors [71]. Below follows a brief introduction to GPU architecture and how different approaches have been used to maximise their efficiency. The abstraction used by the CUDA programming model is also introduced. This allows the same code to be compiled and run on different generations of Nvidia graphics hardware. A basic understanding of both is need before a parallelisation strategy can be chosen and specific algorithms designed. Since most GPUs have relatively similar architectures (at least conceptually), the principles discussed below should be generally applicable to GPUs from many different manufacturers.

The main features of GPUs is their massively parallel nature - while CPUs may have multiple cores (e.g. 2, 4, 8 or occasionally more) it is typical for GPUs to have hundreds or even thousands of cores. In-fact, GPUs do not have independent 'cores' in the same sense that CPUs do. They are classified as Single Instruction stream Multiple Data stream (SIMD) devices, which means each instruction unit issues the same instruction to many compute cores. Each of these cores then performs the same operation in parallel but on different items of data. For computations that have a high degree of parallelism this difference in architecture gives GPUs access to several performance benefits. In direct contrast to CPUs, which maximise performance using high clock speeds, execution optimisation, and large caches, GPUs achieve high aggregate throughput primarily by maximising performance in the areas of:

- Number of transistors dedicated to data processing
- Latency hiding
- Memory access patterns

Firstly, a SIMD architecture contributes to GPUs having higher ratio of transistors dedicated to data processing than CPUs do. For a given size and manufacturing technique there is fixed number of transistors that can fit on a

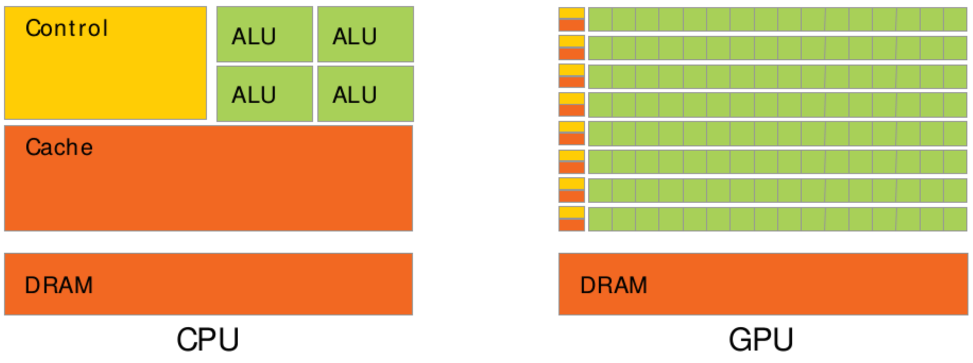


Figure 5.1: GPUs typically have smaller caches and simpler control optimisation. This allows a larger proportion of their transistors to be devoted to the arithmetic logic units (ALUs) used for data processing. [72, fig. 3]

processor chip, so the more of these transistor that are dedicated to caching and flow control the less are dedicated to data processing. A small number of instruction units issuing instructions to many data processing cores gives GPUs a higher ratio of transistors dedicated to data processing than CPUs. This ratio is increased further as GPUs tend to do less execution optimisation and have smaller caches. This is shown schematically in Figure 5.1.

Secondly, GPUs hide latencies by switching to different threads when there is a delay in execution. Like CPUs, any data dependencies or memory latency that causes execution to pause reduces the amount of useful computation that can be carried out. However, in devoting more transistors to data processing GPUs must keep their compute cores supplied with work without the benefit of complex execution optimisations and large caches. They achieve this by having multiple threads resident (i.e. stored in on-chip registers) at any one moment. Since the threads are already in registers, the additional time overhead for consecutive execution of instructions from different threads is very small. This is very different to the case for CPUs where there is a relatively large cost to switching between different threads (known as context switching).

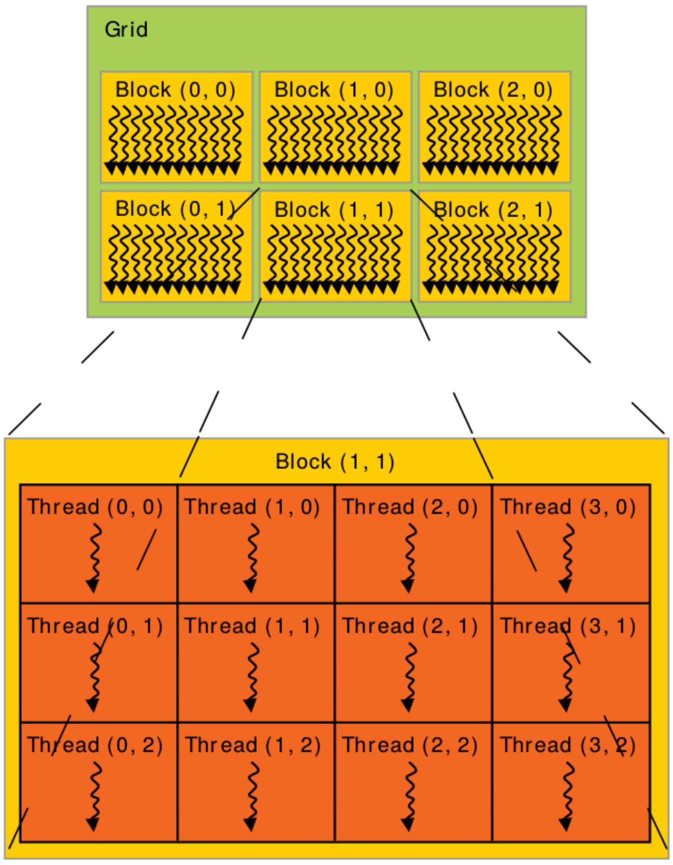


Figure 5.2: The CUDA abstraction organises threads as a grid of blocks. These can be indexed in one-, two- or three-dimensions. [72, fig. 6]

To illustrate this latency hiding for a modern GPU it is helpful to consider CUDA's abstraction and how this relates to the physical hardware. CUDA's abstraction allows SIMD by running a single series of instructions (the *kernel*) on multiple threads organised as a *grid* of *blocks* of threads. This is illustrated in Figure 5.2. Threads within the same block are executed simultaneously and serviced by the same instruction unit. In practice they are not all executed at exactly the same time but batches of 32 threads (known as a *warp*) are all given the exactly the same instruction. The differences in results comes from the fact that the data in the thread registers will be different - at the very least each thread stores a unique ID number of its position with that block. Hardware restrictions mean there is a maximum number of threads per block (currently 1024) so in order to run a larger number of parallel threads CUDA allows a kernel to launch multiple blocks. This is known as a grid, and within a grid the different blocks may be executed in any order. Having a grid of blocks allows the hardware more flexibility when scheduling execution but means the algorithms implemented in the kernel must be independent of the execution order of different blocks. Since the maximum number of blocks per grid is not limited by hardware it can be very large and is currently limited to $2^{31} - 1 = 2,147,483,647$. [72, appendix G] By considering how threads in this hierarchy may be implemented in hardware we can also get an idea of the minimum number of threads needed to hide data dependencies. Typically, each warp (of 32 threads) is processed by a group of 8 physical cores. If the pipeline of each core has a latency of 16 for dependent instructions then we need at least 128 resident threads (4 batches of 32) in order to hide these dependencies without execution optimisation. These threads may all be from the same block but it would be just as valid for them to be from 4 different grids, each containing one block of 32 threads. This flexibility allows scaling across different hardware and also, by having a more active threads, can contribute to hiding the larger latencies of memory access.

Finally, the third technique GPUs exploit to gain performance is the decrease in average latency when threads reading from contiguous memory locations. CUDA refers to the CPU as the *host* and the GPU as the *device*. Both host and device have their own Dynamic Random Access Memory (DRAM) which are addressed separately and usually connected via a PCI or PCIe connection. Host DRAM if often simply referred to as RAM, so to avoid confusion device DRAM is referred to as *global memory*. Similar to RAM access on a CPU, it take a relatively long time for a thread to access data stored in global memory. However, global memory is accessed via 32-, 64- or 128-byte transactions and if threads of the same block access data that is close enough together then a single transaction may serve multiple threads. The extreme case of this is where adjacent threads access adjacent memory addresses allowing full utilisation of the available memory bandwidth. This is known as *coalesced* memory access.

Simulation of train trajectories is compute intensive so (assuming a good parallelisation strategy is devised) the number of floating point operations per second is likely to be a good indicator of the speed of computation. Figure 5.3 compares the *theoretical* speed of Intel CPUs and Nvidia GPUs. As might be expected for hardware designed to maximise aggregate throughput, GPUs excel in this performance metric. As discussed above, the main reason for this is that GPUs devote more transistors to data processing than CPUs do. In striving for low latency on a small number of threads CPUs devote many more of their transistors to complex control optimisation and large caches. In contrast, GPUs hide latencies by switching to different threads when there is a delay in execution. To capitalise on their excellent parallel computation speed,

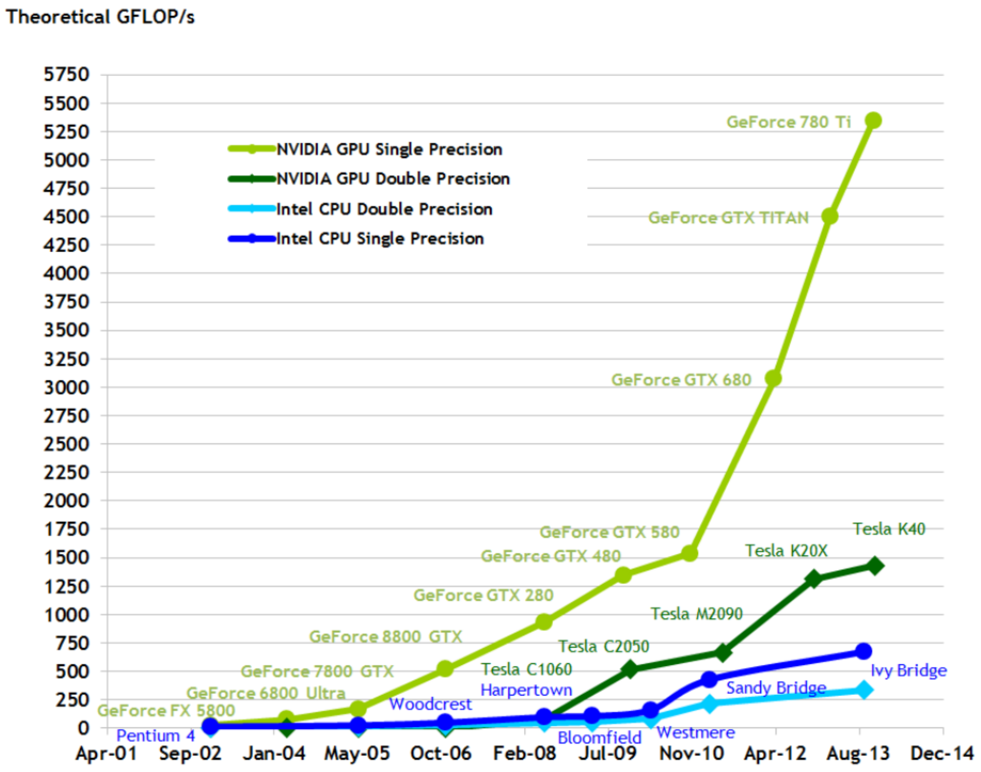


Figure 5.3: Theoretical data processing power of CPUs and GPUs measured in Floating-Point Operations per Second (FLOP/s). [72, fig. 1]

Table 5.1: Approximate latency for GPU threads accessing different types of memory

memory type	**latency /cycles**	**source**
shared	38	[73]
global	∼440	[73]
host RAM (via PCIe)	∼10,000	[74]

which is particularly relevant to multi-train simulation, GPUs were chosen as the target architecture for the new model.

5.2.3 Principles for efficient GPU algorithms

The properties of the GPU architecture discussed in Section 5.2.2 lead to several principles for devising fast algorithms. They can be grouped into three categories [72, chap. 5]:

- Maximise parallel execution
- Optimise memory usage
- Optimise instruction usage

In order to fully utilise the GPU hardware, algorithms must contain as much parallelism as possible. This maximises the chance that all components can be kept busy and so is likely to give the best total performance. This parallelism must also map well on to the system. For example, there must be a sufficient number of threads per block and this number should be a multiple of 32 (the number of threads in each warp). Also, the number of blocks should be large enough that each instructions unit has multiple blocks to switch between allowing it to hide latencies.

The read-latency of different types of memory varies by around three orders of magnitude (see Table 5.1). This means memory usage should be optimised to avoid algorithms becoming limited by memory throughput. Data transfer between host and device has a relatively low bandwidth so the data stored on the host RAM has the highest latency for a thread to access. To address this, memory transfer between host and device should be minimised - both in total size and frequency [75]. Each memory transfer has a relatively large overhead so grouping many small transfers into one larger one will improve performance. Alternatively, if a relatively small amount of computation is taking place on the host between each transfer then it may be that this computation should be moved on to the device. Even if the computation is slightly slower on the device this is likely to be offset by the time gained by removing host-device memory transfer. Finally, even if host-device memory transfers are unavoidable, it may be that they can be overlapped with computations to limit their effect on overall performance. Next, global memory access should be optimised. Again, this means minimising the transfer of data between global memory and on chip memory (e.g. registers, caches, and shared memory). Ideally, all global memory reads should be coalesced to allow full utilisation of the available memory bandwidth (see discussion in Section 5.2.2). Where this is not possible, shared memory may allow algorithms to exploit patterns in memory layout. Shared memory is effectively a user managed cache. If it can be filled using coalesced reads by all threads in a block, scattered reads/writes can then be performed in shared memory making them an order of magnitude faster. Stores to global memory are asynchronous so not as likely to cause stalls directly. However, they use the same memory bus as reads, so will contribute to a loss of performance if this bus becomes saturated.

Finally, when memory throughput is no longer the limiting factor, instruction usage should be optimised. This involves minimising the use of low throughput arithmetic instruction and minimising intra-warp divergence. The normal implementation of transcendental functions such as `sin`, `cos`, or `exp` compile to many native instructions. If their use can not be avoided by algorithm modification, another possibility is to replace them with faster (but less accurate) intrinsic functions. In CUDA this can be done manually or by using the nvcc compiler option (`-use_fast_math`). Also, Nvidia's GTX hardware is primarily targeted at graphics applications so double-precision floating point calculations are significantly slower than their single-precision equivalents. Floating point numbers allow computers to represent a larger range of numbers compared to a fixed point representations using the same number of bits. However, this leads to a loss of precision so the two different sizes of floating point number are commonly used. Single-precision floating point numbers are encoded using 32 bits and, for situations where higher precision is required, a 64 bit (double-precision) floating point can be used. For the latest generation of Nvidia graphics cards built on the Maxwell architecture, including the GTX 750Ti card used in this thesis, the peak rate of double-precision operations is 1/32 that of single-precision operations[76]. The other major instruction optimisation is to minimise intra-warp divergence. All threads in a warp must executed the same instruction so if threads follow different paths through an algorithm then some threads will be inactive until the paths rejoin. This leads to a loss of performance and can be avoided by: minimising the branching algorithms or making sure that all threads in a warp follow the same path.

5.3 Design choices for the GPU accelerated model

These design choices address the limitations of the previous formulation, discussed in Section 5.1.2. Generally, the design choices are described in the order they were made. This means later design choices are dependent on earlier ones. The parallelisation strategy adopted is only possible due to the choice to have soft headway constraints. Likewise, the method for checking headway violations uses the concept of journey features and also assumes soft headway constraints. Both journey feature and the form of the control strategy chosen are designed for efficient execution using the parallelisation strategy adopted.

5.3.1 Soft headway constraints

Constraints are referred to as 'hard' if all solutions are required to satisfy them. Many of the more fundamental limitations of Yang's model stem from its use of hard headway constraints. These include:

- waste of computational resources as all information stored in invalid solutions is discarded
- difficulty performing random initialisation on more complex systems as no information can be passed to successive attempts
- unable to model junctions as it is not possible to check headway constraints through nodes

While hard constraints make sense for the final solution (safe separation *must* be maintained between all trains at all times) they are not necessarily best during the optimisation process. So called 'soft' constraints would allow solutions that breach headway constraints to be penalised in proportion to their infringement, rather than completely discarded. Retaining information stored in invalid solutions enables the optimisation to learn from failed attempts and so make better use of computational resources. This difference in information is represented in Figure 5.4 It is trivial to check the final solution

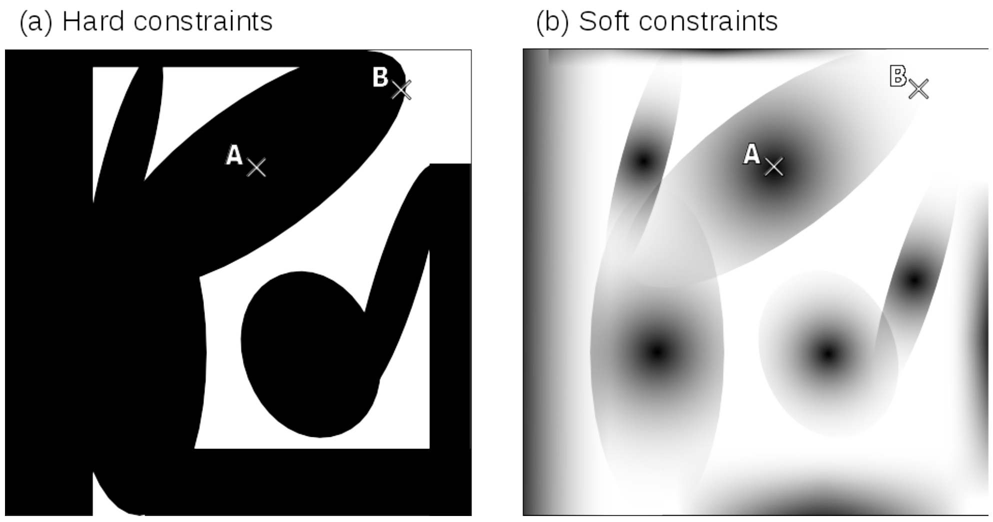


Figure 5.4: (a) and (b) are a 2D representation of the same search space but with hard and soft constraints respectively. In (a) black represents regions where hard constraints are broken (i.e. invalid solutions) so the solutions at points A and B indistinguishable. In (b) areas are shaded in proportion to the extent to which soft constraints are broken (darker points representing larger penalties). Although both points A and B still break constraints it is now clear that point B is less bad than point A.

with hard constraints since any headway penalty will indicate that the constraints were breached in some way. If large penalties are given for breaking constraints then the heuristic optimisations will prioritise keeping these constraints. The use of soft headway constraints has been applied in number of single-train trajectory optimisations Cheng et al. [27], Colin Cole [28].

Soft headway constraints are also likely to aid initialisation in complex systems. In a system where the majority of randomly generated solutions are invalid then a formulation using hard constraints will take many attempts to complete initialisation. If the randomly generated solutions are equally spread over the whole search space then the mean number of attempts needed is the reciprocal of the fraction of valid solutions in the search space multiplied by size of the population being initialised. For example, in Figure 5.4(a) approximately 1/4 of the search space gives valid solutions, so to initialise a population of 100 solutions we would expect to need around 400 attempts.

Additionally, soft headway constraints also allow journey simulation and headway checking to be completely separated. This has three main advantages. Firstly, junctions are now trivial to model as headway constraints can be checked after all journey simulation has occurred. Secondly, it allows a much greater degree of parallelisation to be exploited. Since the boundary conditions of all train journeys are known (each train starts and ends a journey at rest), checking headway constraints separately allows all the simulation of all train journeys to occur in parallel. This fact is exploited when devising a parallelisation strategy in 5.3.2. Finally, it removes the requirement that trains depart stations in a predefined order, allowing some degree of schedule optimisation to occur at the same time as trajectory optimisation. However, since trajectory optimisation will remain the primary purpose of the new model, train routes will still be predetermined. This is important for the method, introduced in 5.3.3, that describes any property of the track that varies with distance.

5.3.2 Parallelisation strategy

Before deciding how to remedy other limitations outlined in Section 5.1.2 it was necessary to choose a parallelisation strategy. The first decision is how to divide the problem into independent tasks that can be executed concurrently (i.e. the order of execution will not affect the result). Next, there should be some level of parallelism within each of these independent tasks. This fits the CUDA abstraction of a grid of thread blocks, where threads within a block are executed in parallel (simultaneously), but different blocks are executed concurrently (independently). Given GPUs are the target architecture, the optimisation principles discussed in Section 5.2.3 must be considered.

Many heuristic optimisations explore the search space using a population of solutions, so evaluating each solution (network control strategy) separately is one way to divide up the problem. However, as discussed above, when simulating each network control strategy it is also possible to divide this problem into the simulation of separate journeys. Considering each different train is another natural way to split up the problem. These options lead to several possible parallelisation strategies, two of which will be discussed here.

Firstly, each network control strategy could be evaluated concurrently. This is likely to be a good abstraction for a CPU as it is easily scalable to many cores and requires no communication between cores except to return the simulation results. On a GPU this leaves the decision of whether trains or journeys should be calculated in parallel. Neither of these seems to lend themselves to a particularly advantageous CUDA thread structure, although a block for each network control strategy containing a thread for each train may offer advantages if hard headway constraints were being enforced. If a time-stepping simulation was used then headway checking could be integrated into the simulation loop allowing trains to react to signals. It is likely that by using shared memory this process could be made relatively fast. However, each thread would require different train data and also read different track data as the simulation progressed. A lot of data would have to be read from global memory (a relatively slow operation) and it is unlikely that reads of track data would be coalesced. All of this would slow simulation. Also, since the number of threads per block is currently limited to 1024 this would limit the modelled system to a maximum of 1024 trains.

Alternatively, each journey could be evaluated concurrently. This leaves the option of simulating the multiple possible trajectories, produced by different control strategies over the same journey, simultaneously. In the CUDA programming model this means within each block the journey is the same for all threads, but each thread uses the control from a different individual in the population. This has the advantage that all threads in each block require the same train data. Also, if a distance-stepping simulation is used, then all threads will require the same track data at the same time (see Section 5.3.3). They can also use coalesced loads to read control points (see Section 5.3.5) and are likely to have low thread divergence. Therefore, this parallelisation strategy was adopted as it looked the most likely to enable fast simulation on a GPU. Since the start time of a journey will depend on any previous journeys made by that train, a timing synchronisation stage will be necessary before headway constraints can be checked. However, this is likely to be a small overhead compared to trajectory simulation.

5.3.3 Pre-calculating journey information

One desirable capability of the new model would be to have a general method for efficiently including any property of the track that varies with distance (e.g. gradients, curves, and tunnels). Since journey simulation is the most computationally intensive part of the optimisation, anything happening inside this main simulation loop must be as efficient as possible. Since the route of each train is not part of the optimisation it is possible to pre-process the data needed during the simulation of each journey. From this point any distance-based event is referred to as a *feature*. If the distance of all features encountered are calculated relative to the start of each journey then they can be stored in a single array in the order they will be encountered. This removes the need for any searching of data inside the simulation loop and also also ensures the next feature is stored in the adjacent memory address (maximising the likelihood of cache hits).

Different types of features can be stored, each offering further opportunities to use pre-calculation to remove calculations from the simulation loop.

Firstly, the gradient profile of the track must be described by using features. However, if it was stored directly then the force on the train would be calculated multiple times during the simulation of each journey. Since this involves the interaction of the mass distribution of the train and the gradient profile this would be a relatively expensive procedure. Relative to distance, the result will be the same for the whole population and will also be constant between generations so repeated calculation does not seem the most efficient approach. Instead, assuming the train length and mass distribution of the train are constant, the component of train acceleration due to the track gradient can be calculated once and stored in the features. This is a similar idea to the result shown by Howlett and Pudney [19], where any control problem

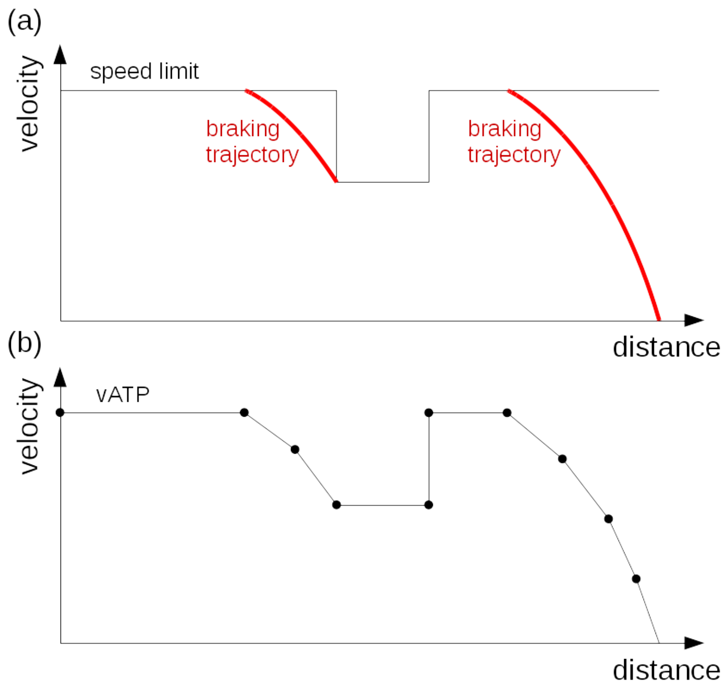


Figure 5.5: The process of converting a line speed limit to an automatic train protection speed limit (vATP) described by a series of features. (a) First, the braking trajectories needed for the train to obey each drop in line speed, and the final stop, are calculated. (b) The speed limit and braking trajectories are approximated as linear functions of distance. Each linear section is described by the feature immediately preceding it. A series of features can then describe the vATP limit over the whole journey (ten features in this example).

for a train with distributed mass has an equivalent point mass problem.

Secondly, the maximum speed limit at any point can be encoded using features. However, similar to the gradient profile, if this is done directly then many calculations will be repeated when finding a suitable brake site at the end of the journey (and for each drop in line speed limit). As discussed in 5.1.2 this is a relatively costly procedure for which no well-defined algorithm was given. An alternative approach, used by Chang and Sim [25], is to pre-calculate speeds as would be used in a simple Automatic Train Protection (ATP) system to prevent line speed violations and bring the train to a stop at its destination. This is not a complete ATP system, as it does not react to the interlocking constraints of the signalling system, so will be referred to a vATP (with the 'v' standing for velocity, to signifying it only enforces the line speed limit). The vATP will be implemented by pre-calculating braking trajectories when there is a line speed drop. These braking trajectories are specific to each train and will be approximated by a polygon curve. The features can then describe a piecewise linear speed profile, where each linear vATP limit is defind by:

$$\text{vATP} = md + c \tag{5.1}$$

where d is the distance from the start of the journey, and m and c are constants. This process is illustrated in Figure 5.5.

Finally, features can encode critical positions used in the implementation of a signalling system. This is a more complex topic and is addressed in Section 5.3.4.

5.3.4 Fixed block signalling

Railway networks use signalling systems to ensure the safe separation between trains is maintained at all times. There are various different types of signalling system, but most modern systems can be classified as fixed block or moving block. Fixed block signalling was implemented in the new model for the reasons outlined below. It should also be noted that signalling blocks are in no way related to the blocks of threads used by the CUDA programming model. Although the word 'block' has different meanings in different disciplines it is fundamental to both railway signalling and CUDA programming. This thesis will use the word in both ways, relying on the context for clarity.

Fixed block signalling systems divide all lines into areas called blocks. Each block is then protected by signals which communicate movement authority to other trains. Safe signal states are ensured by the interlocking between the different blocks. The size, position and interlocking of the blocks are carefully calculated so that a safe braking distance is always maintained between adjacent trains. Figure 5.6(a) illustrates this for three aspect signalling. However, since there is no information about the speed or position of either train within its block the worst cases must be assumed when designing a fixed block signalling system. This means there is usually some 'wasted' capacity, for example when a train is travelling below the maximum permissible speed its braking distance will be shorter than the worst case assumption. Another example is when the lead train is about the leave a signalling block but the following train must still drive as if the lead train is still right at the start of that block. Reducing the size of signal blocks and increasing the number of signal aspect types will allow the trains in both these examples to run closer together, thereby increasing capacity. This is illustrated in Figure 5.6(b). In physical systems there is a limit to the amount of extra capacity that can be

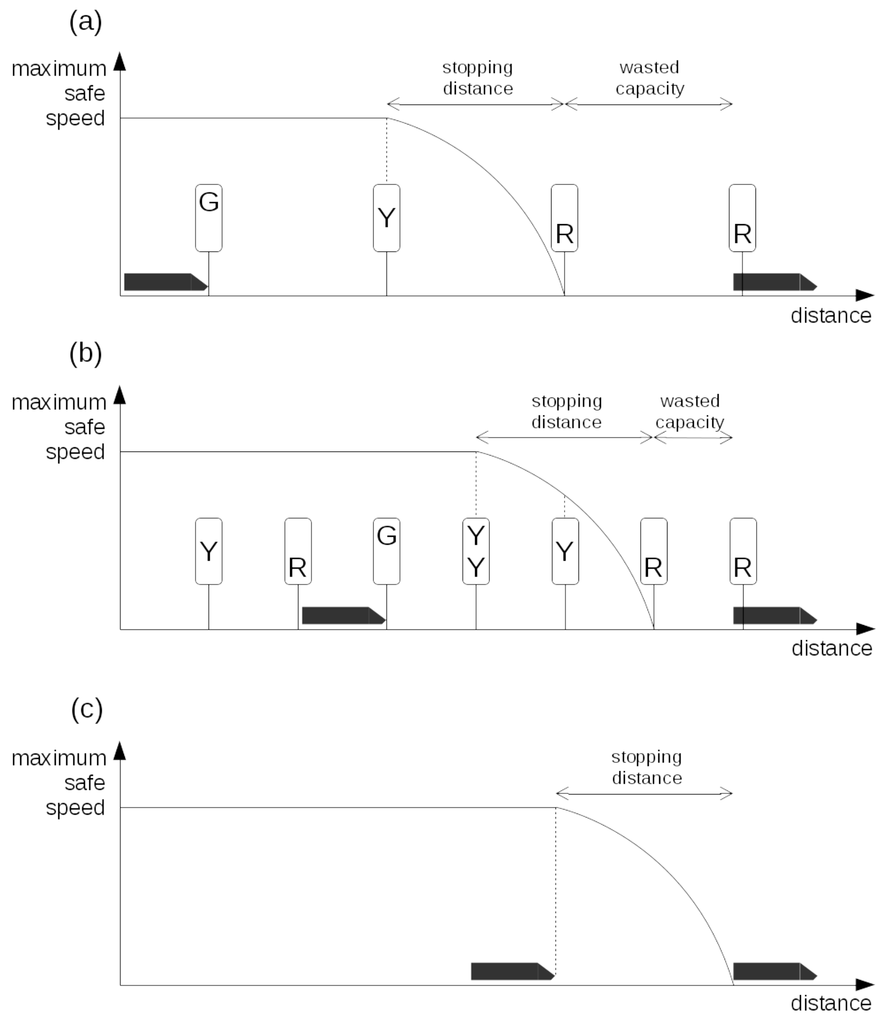


Figure 5.6: Different signalling systems. (a) Three aspect fixed block signalling system. Each block must be at least as long as the train's maximum stopping distance (including some safety margin). Since the lead train may be anywhere in the block the capacity 'wasted' will be up to the length of the stopping distance. In reality other safety measures are usually present, such as signal overlaps or an absolute block of separation, which will both reduce capacity further. (b) Four aspect fixed block signalling system. Each block must be at least half as long as the train's maximum stopping distance. This halves the 'wasted' capacity compared to three aspect signalling but requires twice the signalling and track equipment. (c) Moving block signalling system. The 'wasted' capacity is reduced to a minimum (the limit tended to by n-aspect fixed block signalling).

unlocked this way. Firstly, the number of different aspects that drivers can safely react to is likely to be limited, especially considering the amount of time they have to react to them will also decrease. Secondly, the increased installation and maintenance cost of the additional signalling and track equipment may not be economically justifiable. Moving block systems unlock this capacity by defining blocks relative to the position of each train rather than at fixed locations on the track. The train must regularly update its moving block, which usually consists of the length of the vehicle, its stopping distance and some safety margin. The moving blocks of different trains are then compared to check that they do not overlap. This is illustrated in Figure 5.6(c). Level 3 of the European Rail Traffic Management System (ERTMS) proposes moving block technology at a conceptual level[77], although initial trails used a 'virtual fixed block'[78].

From the perspective of implementing a fast simulation, fixed-block signalling has many advantages over moving block signalling. The size, position and interlocking of blocks are all calculated in advance so during simulation only the occupancy of blocks, and the compatibility of these occupancies, needs to be calculated. In contrast, moving block signalling requires regular calculation of stopping distances. For the proposed model, calculating the exact stopping distance would be a very computationally expensive process. The greater frequency with which blocks must be compared would also add to the computational burden. If/when moving block signalling is required then it can be approximated arbitrarily closely by using very small fixed blocks and a large number of different signal aspects. While this may quickly become infeasible for a physical system it is likely that the new model will achieve a very fine resolution before block comparison becomes limiting.

For these reasons a fixed block signalling system will be implemented in the new model. During journey simulation the speed and time that each train enters or clears a block must be recorded. This allows block events to be stored using the journey features described in Section 5.3.3. The interlocking between different blocks can be defined and different signal aspects defined by specifying speed limits for block exit. This process is described in detail in Section 5.4.5.

5.3.5 Control sequence

The form of control used in the old model turns out to be poorly suited to efficient implementation on a GPU due to the difficulty of ensuring coalesced memory access patterns. Also, as stated earlier, control actions were restricted to traction, coasting and final braking only - restricting the area of the search space that could be described. To overcome these issues a new form for the control strategy is defined in Equations (5.2) to (5.6). Rather than each control point defining the distance of a switch between traction and coasting, each control point becomes a distance-action pair (d_n, c_n). The control action (c_n) is a continuous variable where 0.0 describes coasting, positive values up to +1.0 describe the fraction of maximum traction applied and negative values down to -1.0 describe the fraction of maximum braking applied. This increases the search space that it is possible to describe. The control distance (d_n) is also a continuous variable specifying the position that control is applied. However, to facilitate efficient implementation on GPUs, each control distance (d_n) lies within a fixed range which means the number of control actions ($n_{max} + 1$) is fixed for each journey. A trajectory defined by the new control strategy is illustrated in Figure 5.7.

The number of control points ($n_{max} + 1$) defined for journey j can be

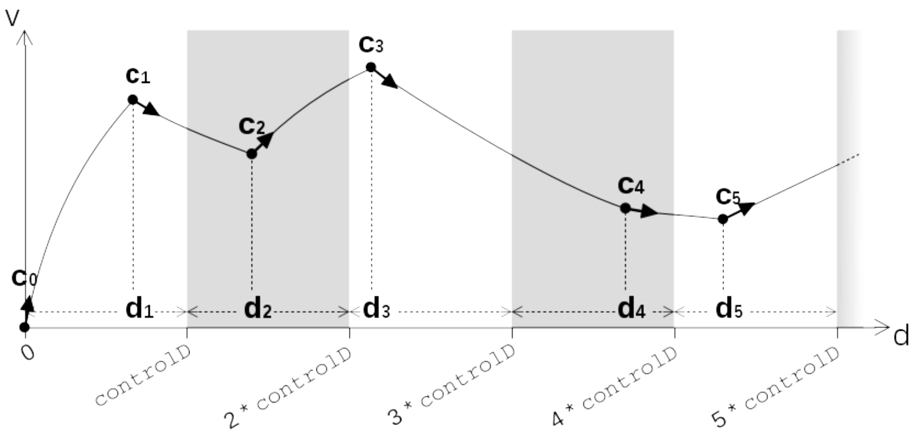


Figure 5.7: A trajectory illustrating the new form of the control strategy. A series of distance ranges are defined (of size `controlD`) each containing a single control point. Every control point specifies both the control action (c_n) to apply and the exact distance (d_n) at which it should be applied.

determined using Equation (5.2).

$$n_{max} = floor(D_j/\texttt{controlD}) + 1 \tag{5.2}$$

where D_j is the length of journey j, `controlD` is the size of the control distance range, and $floor$ rounds down to the nearest integer

Equations (5.3) and (5.4) define the range of values that each control distance (d_n) and control action (c_n) can take.

$$(n-1)\texttt{controlD} \leq d_n < n\texttt{controlD} \tag{5.3}$$

$$-1 \leq c_n \leq 1 \tag{5.4}$$

for $n \in [1, n_{max}]$

However, since the train begins a journey from rest, and most stations are close to horizontal, it is required that the first control action be tractive. It is anticipated that even for a station with a downward gradient some traction will be applied. This is a first control point is special case and is defined in (5.5) and (5.6).

$$d_{n=0} = 0 \tag{5.5}$$

$$0 < c_{n=0} \leq 1 \tag{5.6}$$

By placing these restrictions on the control strategy, coalesced global memory accesses can be guaranteed on the GPU. The parallelisation strategy chosen in Section 5.3.2 means that the trajectories, resulting from different control strategies over the same journey, are simulated in parallel. Since, a distance-stepping model is used, the position along the journey remains in sync between the different simulation threads. If the distance ranges for control points are the same across the whole population of solutions then when the start of a new range is reached all threads can load the next control point at the same time. In contrast, if the position of control points were not restricted by distance ranges then there would be no guarantee that the position of the control points would correlate between different threads. This would lead to fewer coalesced memory transactions and therefore less efficient use of the available memory bandwidth. The actual structure of the data needed for coalesced memory access is discussed more in Section 5.4.2 but is made possible by the properties of the control strategy defined here.

Also, the insertion and deletion operations described in Section 3.3 should no longer be necessary. This is because they were needed to escape local minima in the search - where a large distance separated adjacent control points. Provided the distance range determined by `controlD` is sufficiently small this situation is no longer possible with the new form of control. This is fortunate as the new, fixed length, control strategy is not compatible with insertion and deletion operations in their current form.

Finally, since soft headway constraints have been introduced, station dwell times are no longer determined by whether trains are clear to depart from a station. To some extent the time of departure can be influenced by the control of movement on the previous journey, but this in not adequate on its own. This means the dwell time at stations should also become a control variable in order to better meet target arrival time whilst obeying headway constraints. The minimum departure time may be restricted by a scheduled departure time or simply by the minimum dwell time needed for operations (e.g. boarding and alighting). This makes the model similar to some energy aware schedule optimisation techniques which use dwell and traverse times to control the schedule [58].

5.4 Description of the GPU accelerated simulation

The design choices described in Section 5.3 are implemented in a new GPU accelerated model, which will be referred to at G6. The critical process in the model used in G1 to G5 was simulating train journeys, with over 90% of computation time spent in this function. This meant the journey simulation step was the focus of optimisation efforts when devising G6. The level of detail in the following sections reflects the importance of this journey simulation stage to the overall performance of the multi-train simulation.

In light of the design decisions (described in Section 5.3) the design process proceeded as follows:

- The components of the objective score were defined (headway violations, delays, energy consumption).

- The *content* of the output data from the journey simulation step was determined by inputs required by the subsequent steps.
- The *content* of the input data to the journey simulation step was determined by its required outputs.
- The *structure* (i.e. memory layout) of both the input and output data for journey simulation was designed alongside its algorithms to ensure good memory access patterns.
- The other algorithms and data structures were then designed to fit in with the journey simulation data structures.

As such, the importance of the data structures devised cannot be over emphasised. In order to fully grasp the significance of the algorithms discussed below, one must always keep in mind the data structures that are being read from or written to.

5.4.1 Overall structure

When considered at the highest level of abstraction the optimisation can be split into two parts (i) simulation and (ii) the optimisation algorithms themselves. These are show in Figure 5.8.

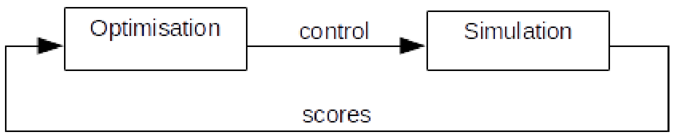


Figure 5.8: The two alternating stages of the model - simulation and optimisation.

The lowest available bandwidth is between the host and device so host-device data transfer has the greatest potential for negatively impacting performance. To avoid this all parts of both simulation and optimisation were implemented on the GPU. This was a significant undertaking but meant that host-device data transfer only had to happen at the start and end of the whole optimisation so did not cause a significant overhead.

Given the design decisions in Section 5.3, the simulation stage must act on a whole population of control strategies at once. It takes a population of control strategies, performs phenotype evaluation, and then outputs a score for each member of the population. The cost function depends on three factors: total energy consumption, total delay (time after scheduled arrival), and headway violations (determined by the number and severity of signal block incompatibilities). Each of these leads to a distinct stage during phenotype evaluation before scoring can occur, These stages are illustrated in Figure 5.9.

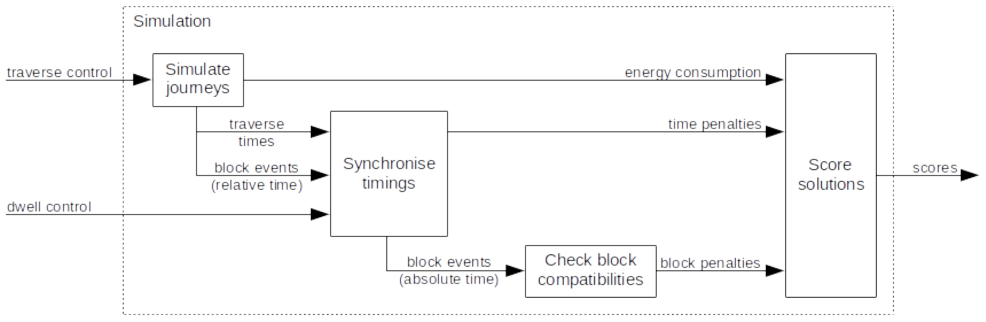


Figure 5.9: The four stages of simulation are designed to allow efficient computation on a GPU. The algorithms behind each stage are described in Sections 5.4.3 to 5.4.6.

First, the movement of each train over each of its journeys is simulated. This gives the traction energy consumption, the time taken to traverse each journey, and the relative times the train entered or left signalling blocks that were encountered. The traction energy consumption it not dependent on any other journey so does not require any further processing. However, the other two data structures require further processing before they can be compared with either the schedule or block occupancies from other journeys. Second, timings are synchronised between all journeys with each solution. Thirdly, the times at which each signal block is occupied must be compared to ensure that safe operational headways are maintained between all trains. Finally, the three different penalties can be added up for each solution and then combined to give an overall score.

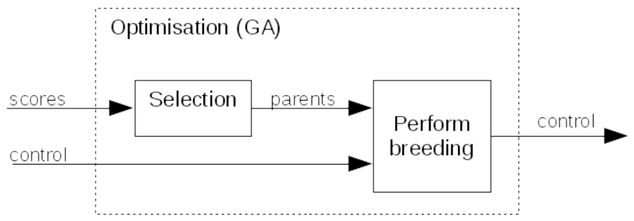


Figure 5.10: A genetic algorithm was implemented as the optimisation algorithm since it is easily implemented on a GPU. The two stages of the GA are illustrated here and described in Sections 5.5.1 and 5.5.2.

5.4.2 Data structures

As discussed in Section 5.2.3 minimising memory latency is crucial for efficient GPU algorithms. Since host-device data transfer has already been minimised (by choosing to implement both simulation and optimisation on the GPU), efficiently accessing global memory within the device now becomes the primary objective. The efficiency with which this takes place is not only determined by the algorithms processing the data but crucially also by the memory layout of the data itself.

Coalesced memory access requires that consecutive memory locations are accessed by consecutive thread IDs. The C programming language guarantees to store arrays in a contiguous block of memory, so for a 1D array consecutive thread IDs must access consecutive array indexes. The situation is slightly more complex for 2D arrays since they are still stored in one contiguous block of memory. To arrange a 2D array in 1D memory C uses row-major order which means consecutive elements of each row are stored in contiguous memory. Where a kernel accesses a 2D array coalesced access can occur if consecutive thread IDs access consecutive columns in the same row. Because of the parallelisation strategy adopted (see Section 5.3.2) all 2D arrays have thread IDs determined by the population number (pID) and block IDs determined by the journey number (jID). This means data is structured so that the row and column indices are jID and pID respectively. The benefit of this is illustrated in Figure 5.11.

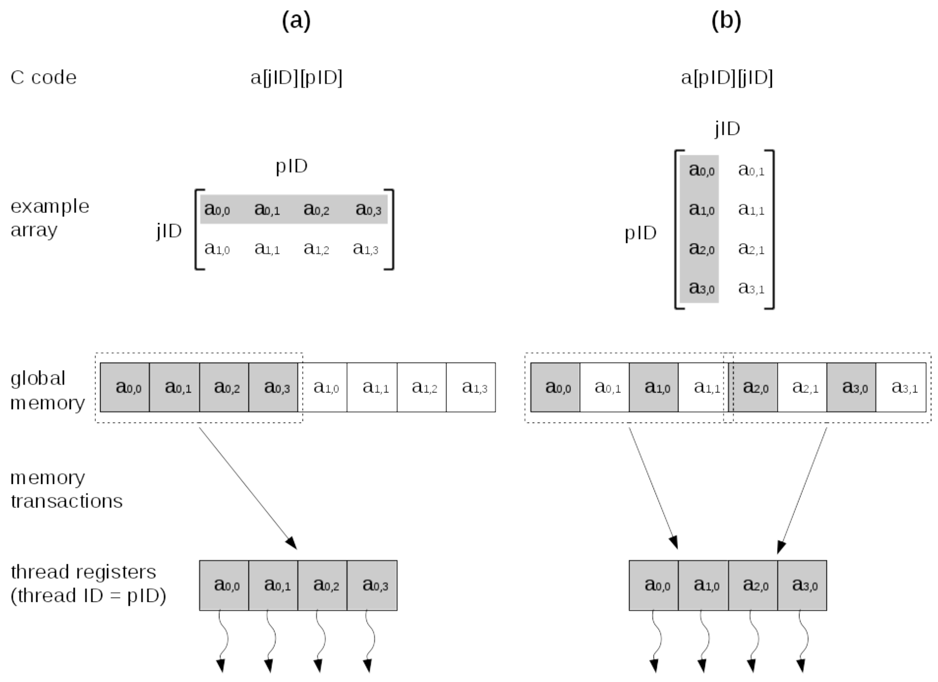
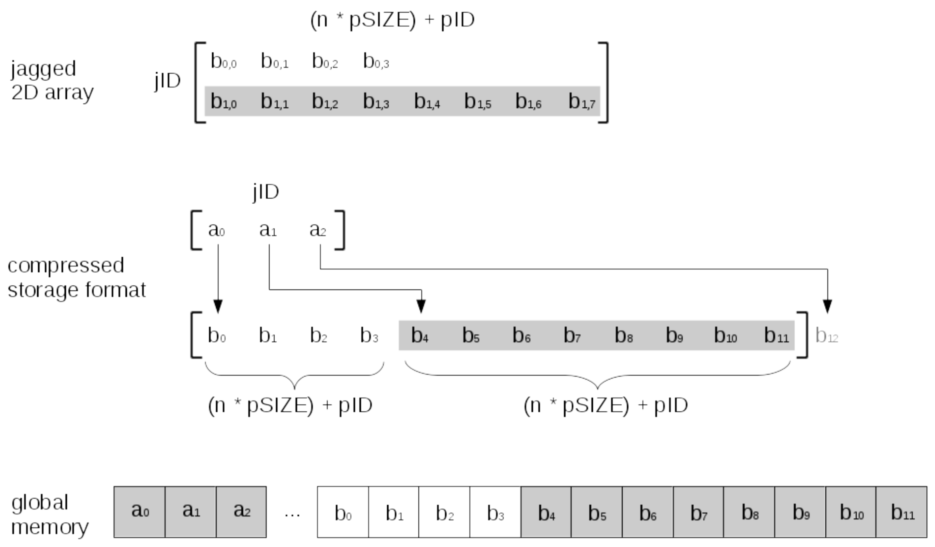


Figure 5.11: (a) Coalesced memory access makes full use of each memory transaction. (b) The transpose of the same data array causes a strided memory access pattern. In this case the memory bandwidth is halved but a larger number of columns would reduce this further.

There are also a number of situations where the amount of data to be stored varies between journeys. For example, a longer journey will require more control actions than a shorter one or different journeys may contain a different number of signal block events. There are two approaches to structuring this data that allow coalesced memory access: to pad the data so that all rows are the same length, or store it in a compressed storage format. The padding approach is simpler whereas the compressed approach is more memory efficient in most cases but introduces an additional level of indirection. However, this additional lookup only takes place once at the start of each journey so it is likely to result in a relatively small overhead. GPUs don't have expandable memory so this memory could become a limitation if complex networks are modelled or large population sizes used. For this reason the more memory efficient compressed storage format was chosen, and implemented as an index lookup table and a 1D array. This is illustrated in Figure 5.12.



(n * pSIZE) + pID
jagged 2D array
jID
b0,0 b0,1 b0,2 b0,3
b1,0 b1,1 b1,2 b1,3 b1,4 b1,5 b1,6 b1,7
jID
a0 a1 a2
compressed storage format
b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12
(n * pSIZE) + pID
(n * pSIZE) + pID
global memory
a0 a1 a2 ... b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11

Figure 5.12: The compressed data storage format used to encode 2D arrays with rows of variable length (sometimes referred to as a 'jagged' array). The data is compressed into a 1D array and a second 1D array is used to look up the start location of each row within the first array. This minimises the memory used but still allows coalesced memory access.

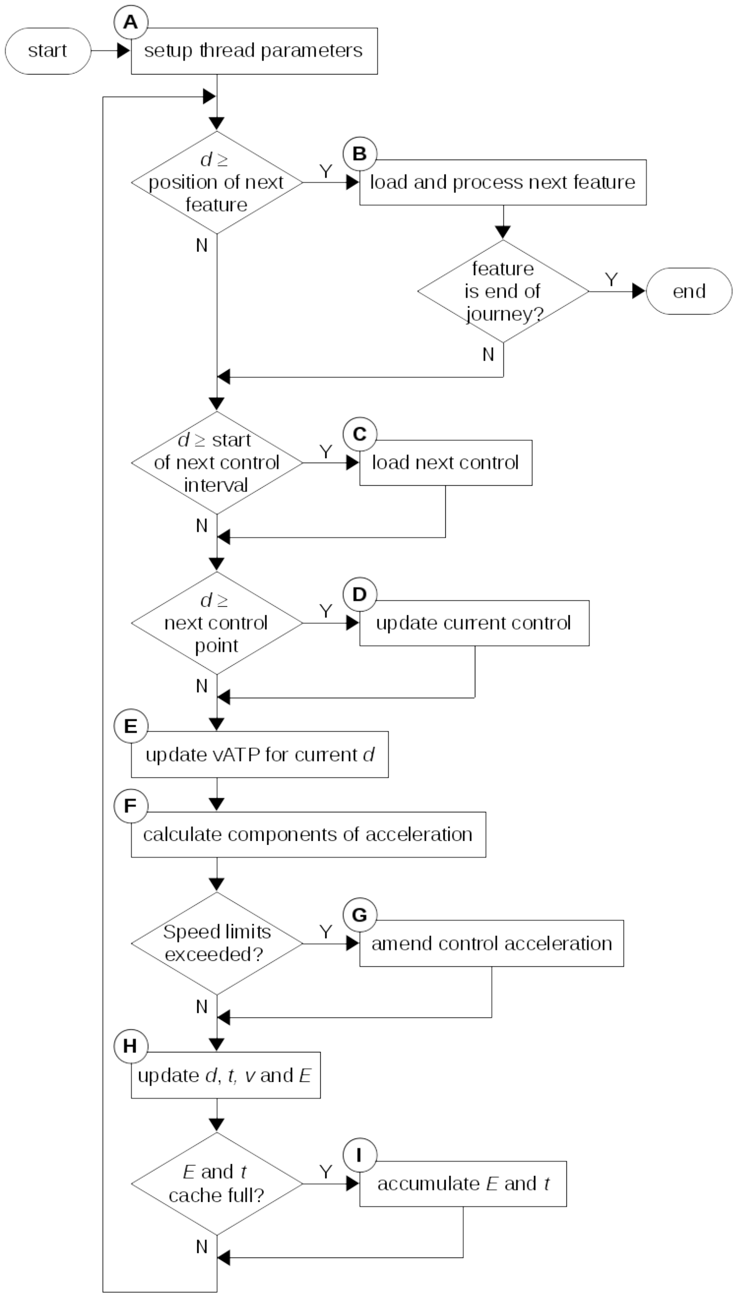
5.4.3 Journey simulation

Journey simulation was the most computationally expensive part of evaluation. Because of this particular attention was made to its optimisation. Due the importance clearly communicating how arrays are indexed, the detailed algorithms described in Section 5.4.3 are give in CUDA/C++ code. This also allow important detail, such as the data types used for different variables, to be communicated in a standard way.

An overview of the journey simulation algorithm is show in Figure 5.13. This was implemented as a single CUDA kernel, the source code of which is given below. Sections of CUDA/C++ code are denoted by a box containing a `monospaced font`. For clarity (and ease of comparison to Figure 5.13) important aspects of this algorithm are discussed under Sections A to I. The order of the kernel code is preserved.

A Set-up the thread parameters

CUDA uses the `__global__` declaration specifier for kernel functions. This function is called from the host but run on the device. The majority of the function parameters are inputs with output only possible to the `outT`, `outB` and `penE` data structures. One might expect `RNGStatesSim` to be changed since the random number generator (RNG) will give the same sequence of pseudo-random numbers if its state is not updated. However, since the RNG is only used to introduce random errors in train parameters it is desirable to get the same result for each train (each `tID`) across all journeys within the same solution (same `pID`). In order for these instances to vary 'randomly' with each successive population the RNG states must be updated once per generation after all journeys have been simulated. This takes place at the start of time synchronisation (see Section 5.4.4). As discussed in Section 5.3.2, tak-



start
A
setup thread parameters
d ≥ position of next feature
Y
N
B
load and process next feature
feature is end of journey?
Y
N
end
d ≥ start of next control interval
Y
N
C
load next control
d ≥ next control point
Y
N
D
update current control
E
update vATP for current d
F
calculate components of acceleration
Speed limits exceeded?
Y
N
G
amend control acceleration
H
update d, t, v and E
E and t cache full?
Y
N
I
accumulate E and t

Figure 5.13: The algorithm used to simulate train movement over each journey. Here, d, t, v, and E are used as abbreviations for distance, time, velocity, and energy respectively. Detailed description of labelled stages A to I are given in Section 5.4.3.

ing distance-steps allows memory coalescing when loading track features and control action. The size of the distance-step (Δd) is defined by `deltaD` so changing this variable controls the balance between the speed and accuracy of the simulation. A small `deltaD` will mean more distance-steps are taken leading to a more time consuming but more accurate simulation. Conversely, a simulation using a large `deltaD` will execute faster but at the cost of reduced accuracy. A sensitivity analysis into the effect of `deltaD` on simulation accuracy is performed in Section 5.6.

```
// define CUDA kernel
__global__ void simJ (
   float deltaD, unsigned int pSIZE,
   float controlD, bool with_noise,
   const control_class* allControl,
   const unsigned int* allC_from_jID,
   const feature_class* allFeatures,
   const unsigned int* allF_from_jID,
   const train_class* allTrains,
   const train_class* allTrainErrors,
   const curandStatePhilox4_32_10_t* RNGStatesSim,
   const unsigned int* outB_from_jID,
   float* outT, block_occ_class* outB, float* penE)
{
```

In contrast to normal C functions, where multiple calls to a function must take place sequentially, CUDA kernels are launched as a grid of thread blocks. This allows a kernel to execute multiple times in parallel and was described in Section 5.2.2. Each thread has two IDs, each of which may contain 3-

dimensions (denoted x, y and z). The IDs are defined by the thread's position in a block (threadIdx) and that block's position in the grid (blockIdx). Within the kernel, these IDs must then be mapped on to the problem being solved. As discussed in Section 5.4.2 it is important that consecutive thread IDs have consecutive pID as this allows coalesced memory access. However, threadIdx is currently limited to 1024 values so in order to enable larger population sizes, pID is also split across blocks. The number of threads per block is given by TILE_SIZE_pop (128 threads per block was found to give the best performance).

```
    // Map launch configuration on to problem
    unsigned int jID = blockIdx.x;
    unsigned int pID = blockIdx.y * TILE_SIZE_pop +
                       threadIdx.x;
    if (pID >= pSIZE)
       return;
```

Since each journey is simulated independently of the others, the distance the train has travelled (d), total time elapsed (totalT) and the total traction energy consumption (totalE) are all calculated relative to the start of that journey. This means they are all initialised to zero. Importantly, the choice of data types for these accumulators was also considered carefully. Single-precision floating-point numbers (32 bits in size) only have a precision of 7 decimal digits [79]. If the accumulator has a value more than 10^6 times larger than the value to be added to it then the result may be rounded to the same value held initially. If this takes place many times then the end result is likely to be significantly different to the expected value. A simple solution to this would be using double-precision floating-points to store all accumulators, since they have 16 digits of precision so are far less likely to encounter pathologic rounding errors. However, as might be expected on GTX hardware (see 5.2.3) performing multiple double-precision arithmetic operations caused a significant increase in kernel execution time.

For the traverse time of a journey:

$$\text{largest possible total time} = D/v_{min} \tag{5.7}$$

$$\text{smallest possible time increment} = \Delta d/v_{max} \tag{5.8}$$

where D is the journey length, Δd is the distance-step, and v_{max} and v_{min} are the maximum and minimum velocities respectively. If a single-precision floating-point variable was used to store the accumulated time then Equations 5.7 and 5.8 can be combined to give the safe usage limits for a single-precision accumulator (Equation 5.9).

$$\frac{D}{\Delta d}\frac{v_{max}}{v_{min}} < 10^6 \tag{5.9}$$

In the new simulation $v_{min} = 0.1$ m/s so simulation of a high speed train with a $v_{max} = 100$ m/s would be limited by $D/\Delta d < 10^3$. It is reasonable to expect users may wish to model situations close to this limit. For example, a 100 km journey with a resolution of $\Delta d = 10$ m would not even guarantee one decimal digit of precision from each distance-step. Whether this is an acceptable resolution is discussed in Section 5.6 but at the very least it demonstrates that using a single-precision floating-point accumulator places an unrealistic burden on the user to check each case modelled.

To overcome this problem double-precision time and energy accumulators were used (`totalT` and `totalE` respectively) but both were 'cached' via single-precision floating-point variables. This process is detailed in stage I but its effect is that for most distance-steps only single-precision floating-point operations occur but the safe usage limits are now given by:

$$\frac{nv_{max}}{v_{min}} < 10^6 \tag{5.10}$$

$$\frac{D}{n\Delta d}\frac{v_{max}}{v_{min}} < 10^{15} \tag{5.11}$$

where n is the number of values cached before the accumulator is updated. Given $n = 50$ was used in the simulation an extreme use case ($D = 10,000$ km, $\Delta d = 0.1$ m, $v_{max} = 100$ m/s, $v_{min} = 0.1$ m/s) still yields at least one decimal digit of precision from each distance-step.

In contrast, the decision was made to use a single-precision floating-point variable as the total distance accumulator (`d`). Unlike, the energy and time accumulators, the values of which are only retrieved when recording signal block events or at the end of the journey, the distance is used every simulation step taken. This removes the possibility of caching via a single-precision floating-point variable. However, since the size of each distance-step and the length of the longest journey are both known in advance they can be compared to check for potential rounding errors. The longest journey simulated must be no greater than 10^6 times the smallest significant decimal digit of the distance-step (Δd). For example, Δd cannot be less than 1 m when the longest journey is 1000 km. This seems a reasonable level of accuracy whilst still allowing most rail networks to be modelled.

```
    // Define accumulators
    float d = 0.0f;
    double totalT = 0.0;
    double totalE = 0.0;
    float cachedT = 0.0f;
    float cachedE = 0.0f;
    unsigned int cached_counter = 0;
```

Where compressed data structures are used, one additional memory read is required in order to find the index of the specific journey being simulated. Since these memory reads take place outside the simulation loop they are only executed once and therefore unlikely to significantly impact the overall performance of the kernel. Also, the ID of the train being simulated (`tID`) is stored in the first element of the journey features so must be looked up before the train parameters can be loaded. This works well as the journey features are pre-calculated and are specific to both the properties of the train and the route it takes. There is also the option to introduce noise in the train parameters. This is intended to allow robust optimisation using the technique described in Chapter 4.

```
    // Lookup first element of compressed data structures
    const feature_class * nextF = allFeatures +
                                  allF_from_jID[jID];
    const control_class * nextC = allControl + pID +
                                  allC_from_jID[jID] * pSIZE;
    block_occ_class * nextB = outB + pID +
                              outB_from_jID[jID] * pSIZE;
```

```
// Load train parameters
const unsigned int trainID = nextF -> get_id();
++nextF;
float nextF_d = nextF -> get_d();
train_class thisR = allTrains[trainID];

// Introduce random error in instance of train parameters
if (with_noise)
{
   curandStatePhilox4_32_10_t local_curandState =
                     RNGStatesSim[trainID * pSIZE + pID];
   train_class this_error = allTrainErrors[trainID];
   thisR.add_error(&this_error, &local_curandState);
}
```

Before entering the main simulation loop it is necessary to declare the variables that will be used. The value of the first control action must be retrieved from memory but initial values of the other variables either don't matter or are given sensible default values. It is important that `v` and `v_old` are not initialised to zero as this may cause undefined behaviour (divide by zero). Instead, they are initialised to a very small positive number. Unlike the other variables the line speed limit and acceleration due to the gradient (`vATP` and `aGrad` respectively) are defined as linear functions of `d` in line with the implicit problem formulation, Section 5.1.1.

```
    // Initialise control
    float c = nextC -> get_c();
    float d_read_nextC = 0.0f;
    nextC += pSIZE;
    float c_bufferC = c, d_bufferC = FLT_MAX;

    // Initialise physics varaibles and limits
    float v = FLT_MIN, v_old = FLT_MIN;
    float v_max_new = FLT_MAX, v_max_old, v_max;
    float a, a_cmin, a_c, a_vmin, a_vmax;
    float deltaT = 0.0f;
    linear_eq_class vATP(0.0f,0.0f), aGrad(0.0f,0.0f);
```

B Load and process the next feature

On entering the main simulation loop, it is first necessary to check for journey features and process them accordingly. Journey features are pre-calculated for a specific train traversing a specific length of track. Each feature is then listed in an array ordered by their distance from the start of the journey. By taking distance-steps during simulation, intra-warp divergence is avoided here, which mean all threads in a warp will follow the same execution branch. This is particularly important as each thread must load the next feature from global memory - a potentially time-consuming instruction. However, since all threads load the same feature at the same time, compute 2.0 devices (and later) will only load this value once and then broadcast it to all the threads using L1 cache [80, p115].

Before loading the next journey feature the cached accumulators are updated as the total time must be up to date. Once the new journey feature has been loaded it must be processed according to the type of data it stores. This method allows each journey to be described using a single array containing different types of data. It is hoped that this will allow easy extension of the simulation to incorporate different new types of journey features (e.g. tunnels, electrical sections, neutral sections, track curvature, etc). The types of features implemented are:

- `FEATURE_vATP`: the line speed (maximum permissible velocity) to be enforced during simulation. This is analogous in function to Automatic Train Protection (ATP) systems, which prevent the driver accidentally exceeding speed restrictions. Each feature gives the gradient (m) and intercept (c) of the linear function ($v = md + c$). Over the whole journey these features are described by a continuous piecewise linear function of the line speed limit. No step discontinuities are allowed between consecutive linear pieces. This guarantees that either `v_max_old` or `v_max_new` will store the lowest speed restriction passed while the distance-step was being taken, preventing the algorithm in stage G missing brief drops in line speed.

- `FEATURE_aGrad`: the component of train acceleration due to the track gradient. When pre-calculating journey features the train length and mass distribution of the train are assumed to be constant. This allows the net force on the train to be calculated at each point along the line. Since the train's mass is constant the acceleration at each point is stored as this reduces the calculation during simulation. Again, each feature stores the gradient (m) and intercept (c) of a linear function (this time, $a = md + c$).

- `FEATURE_block_entry`: the position at which the train enters a new signal block. When this is encountered the signalling block ID (bID), the train ID (tID), and the speed and time at which the train enters the block are all saved. This information is needed for headway checking, detailed in Section 5.4.5. The sign of the speed is used to encode whether the train is entering or leaving a signalling block (-ve for entry, +ve for exit). Also, since the exact position of the block entry will always be passed while taking the last distance-step, both the time and speed are rounded to the worst case values. It should be noted that the signalling block ID referred to here is unrelated to the block ID used by the CUDA programming model.

- FEATURE_block_exit: the position at which the train clears a signal block. (See description of block entry)

- FEATURE_end: signifies the end of the journey. Until this point totalE has actually stored the sum of the acceleration due to traction force over all distance-steps. Since distance steps are of a constant size, multiplication by mass and distance only happens once at the end of each journey rather than once for every distance-step taken.

Finally, the next feature is retrieved and its position checked against the current train position. This allows multiple features to be at the same position on the journey and still be processed correctly.

```
    while (true) {
       while (d >= nextF_d) {
          // Update accumulators
          cached_counter = 0;
          totalT += cachedT;
```

```
cachedT = 0.0f;
totalE += cachedE;
cachedE = 0.0f;
// Process journey feature
switch (nextF -> get_type())
{
case FEATURE_vATP:
   vATP.set(nextF -> get_m(), nextF->get_c());
   v_max_old = fminf(v_max_old, vATP.f(nextF_d));
   break;
case FEATURE_aGrad:
   aGrad.set(nextF -> get_m(), nextF -> get_m());
   break;
case FEATURE_block_entry:
   nextB -> set(nextF -> get_id(), trainID,
             -fmaxf(v, v_old), totalT-deltaT);
   nextB += pSIZE;
   break;
case FEATURE_block_exit:
   nextB -> set(nextF -> get_id(), trainID,
             fmaxf(v, v_old), totalT);
   nextB += pSIZE;
   break;
case FEATURE_end:
   outT[jID * pSIZE + pID] = totalT;
   penE[jID * pSIZE + pID] =
                      totalE * thisR.get_mass() * deltaD +
                      totalT * thisR.get_hotelPow();
              return;
           }
           ++nextF;
           nextF_d = nextF -> get_d();
        }
```

C Load the next control point

Instead of each thread loading the next control action individually, at the point it will be applied, all the threads load their next control action at the same position and then hold it in a register until the point when it must be applied. This requires that all control sequences for a given journey are the same length (though they may vary between different journeys) and that each control point lies within a predefined distance range. A more detailed discussion of the control sequence is given in Section 5.3.5 . This is another example of how taking distance-steps during simulation prevents thread divergence and enables coalesced global memory access. The while loop is necessary to correctly update control in the case that the control point lies within `deltaD` of the start of the next distance range or `deltaD` $>$ `controlD`.

```
        while (d >= d_read_nextC)
        {
           c = c_bufferC;
           d_bufferC = nextC -> get_d();
           c_bufferC = nextC -> get_c();
           nextC += pSIZE;
           d_bufferC += controlD;
        }
```

D Update the current control action

Since the distance at which each new control action is defined will vary between each individual in the population, it is likely that some threads will execute these instructions while others will be left idle. This intra-warp divergence has a negative impact on the overall performance but since this branch contains very few instructions the overall effect is likely to be very small. Also, notice stages B, C and D are all within conditional statements based on distance so are only executed intermittently, as the train reaches a specific distance along the journey.

```
        if (d >= d_bufferC)
        {
           c = c_bufferC;
           d_bufferC = FLT_MAX;
        }
```

E Update speed restrictions

Before the next distance-step can be simulated the speed restrictions must be calculated for the current position. Ideally, the trajectory would be checked against the exact profile of the speed limit but this would be relatively time consuming. Instead, the most restrictive speed limit passed (during the current distance-step) is used when implementing vATP. This is sufficient for safe operation and will follow the exact profile of the speed limit more closely as smaller distance-steps (Δd) are taken.

```
        v_max_old = v_max_new;
        v_max_new = vATP.f (d + deltaD);
        v_max = fminf (v_max_old, v_max_new);
```

F Calculate components of acceleration

Components of each train's acceleration are grouped into two variables: acceleration due to traction or braking (`a_c`), and acceleration due to external factors like resistance and gradients (`a`). The current control action (`c`) determines the initial value of `a_c` which will start off as some fraction of either the maximum acceleration (resulting from maximum traction) or the minimum acceleration (resulting from maximum service braking).

```
        a = aGrad.f(d) - thisR.get_resistanceAcc(v);
        a_cmin = -thisR.get_brakingAcc(v);
        if (c > 0.0f)
           a_c = c * thisR.get_tractionAcc(v);
        else
           a_c = -c * a_cmin;
```

G Amend control acceleration

The train must obey the speed limits so the control accelerations needed to maintain these are calculated (`a_vmax` and `v_vmin`). As well as the maximum velocity, a minimum velocity was also defined (`V_MIN` = 0.1 m/s) to prevent non-positive velocities occurring and causing divide by zero errors or erroneous results. These limits are then enforced by modifying the control acceleration as necessary. The maximum speed limit is treated as 'soft' limit so is only kept as far as realistic traction limits allow. This means the maximum speed limit (defined by the `FEATURE_vATP` features) should include a safety tolerance as discussed in Section 5.3.3. Where a line speed is reduced `a_vmax` should low enough that the train can apply maximum service braking to meet this speed restriction. In contrast, the minimum speed limit is enforced as a 'hard' limit as the model cannot handle non-positive velocities. The implicit assumption here is that the train will always be able to produce the required traction force (i.e. `a_cmax` $\leq$ `a_vmin`). This will be an acceptable assumption for most passenger trains but may be problematic for freight, where a lower power to weight ratio makes getting stuck on a steep incline a realistic possibility. A solution to this would be to use the journey features in step B to define a variable minimum speed limit which could be pre-calculated to include a safety tolerance. Finally, once the control acceleration has been checked (and possibly corrected) the net acceleration of the train can be calculated.

```
        a_vmax = (v_max * v_max - v * v)/(2.0f * deltaD) - a;
        a_vmin = (V_MIN * V_MIN - v * v)/(2.0f * deltaD) - a;
        if (a_c > a_vmax)
           a_c = (a_vmax < a_cmin) ? a_cmin : a_vmax;
        if (a_c < a_vmin)
           a_c = a_vmin;
        a += a_c;
```

H Take distance-step

The new speed and time of the train are calculated by assuming linear acceleration over a small distance-step (`deltaD`, referred to elsewhere as Δd). The traction (or regeneration) efficiency may vary with velocity so, as with the other functions of v, the change in total energy consumption is calculated before `v` is updated.

```
        if (a_c > 0.0f)
           cachedE += a_c / thisR.get_tractionEff(v);
        else
           cachedE += a_c * thisR.get_regenEff(v);
        d += deltaD;
        v_old = v;
        v = sqrt(v * v + 2.0f * a * deltaD);
        deltaT = (2.0f * deltaD) / (v + v_old);
        cachedT += deltaT;
```

I Update cached accumulators

As discussed in step A a good balance of performance and accuracy is achieved by using single-precision floating-point variables (`t_cached` and `e_cached`) to cache changes in the *total* energy and *total* time accumulators (`totalE` and `totalT` respectively - both double-precision floating-point variables). However, this method requires the cache variables to be regularly added to the totals and then be reset. This is implemented using a counter (`cached_counter`), which is incremented once every distance-step simulated. Usually (49 out of every 50 cycles) the accumulators will not be updated, minimising the total number of double-precision floating point operations that must be performed. The simulation loop is then run again, continuing until a `FEATURE_end` journey feature is encountered.

```
        ++cached_counter;
        if (cached_counter >= 50) {
           cached_counter = 0;
           totalT += cachedT;
```

```
            cachedT = 0.0f;

            totalE += cachedE;

            cachedE = 0.0f;

        }

    }

}
```

5.4.4 Timing synchronisation

The parallelisation strategy chosen in Section 5.3.2 means that all journeys are simulated independently of one another. As a direct consequence of this, all times output from the journey simulation stage are given relative to the departure time at the start of each journey. However, one train may be scheduled to make a number of stops and, since a journey is defined as the motion of a train between two consecutive stops, the true departure time is dependent on the traverse times of any preceding journeys that train has made and the dwell times at each station stop. This means that before train punctuality can be calculated, the timings for all journeys must be synchronised with each other. Timing synchronisation is also necessary for any data recorded due to journey features - this means signal block entry and exits times must be synchronised to a single, network wide, time frame. This involves adding the true departure time to the time that has been recorded for each block event.

First, the true departure times are calculated for each journey. As always, when devising an efficient GPU algorithm to do this, it is important to consider global memory access patterns. All data output from journey simulation uses the index within the population (`pID`) as the column index for each array. This means, when choosing a parallelisation strategy for timing synchronisation, it makes sense to have each thread operating on the times from a different pID. The CUDA abstraction requires that these threads are organised into thread blocks (which are not to be confused with signalling blocks used in interlocking). Each thread block must executed independently of the others so, since each arrival time only depends on the preceding journeys made by that train, one thread block is used for each train (indexed by tID). Consecutive journeys of each train are indexed by consecutive jIDs, which allows a look-up array (jiD_from_tID) to be used to convert from tID to jID. This is similar to the compressed data storage format illustrated in Figure 5.12, but with an extra dimension (each array element contains another array indexed by pID).

```
// define CUDA kernal
__global__ void syncT (
   unsigned int pSIZE,
   const unsigned int* jID_from_tID,
   const schedule_class* allS,
   const float* allDwells,
   const float* outT,
   float* depT,
   float* penT,
   curandStatePhilox4_32_10_t* RNGStatesSim,
   bool with_noise)
{
   // Map launch configuration on to problem
   unsigned int tID = blockIdx.x;
   unsigned int pID = blockIdx.y * TILE_SIZE_pop + threadIdx.x;
   if (pID >= pSIZE)
      return;
```

```
    // look-up jID using tID
    unsigned int jID = jID_from_tID[tID];
    unsigned int jID_end = jID_from_tID[tID + 1];
```

Next, the other variables used in the kernel are defined. Since `t` is an accumulator, a double precision floating point data type is used to minimise the impact of rounding errors. The random number state is only loaded if randomly varying the dwell time (giving a G6 the potential for using robust optimisations of the type described in Chapter 4).

```
    // define thread variables
    double t = -FLT_MAX;
    schedule_class tempS;
    float dwell_time;
    curandStatePhilox4_32_10_t local_curandState;
    if (with_noise)
       local_curandState = RNGStatesSim[tID * pSIZE + pID];
```

Finally, a loop is run within the kernel, iterating over all the journeys that train makes, in the sequence with which they are scheduled (`allS`). The dwell time is calculated and added to the arrival time. This yields the departure time for the start of the current journey, which is saved to a global array (`depT`) with the same format as the traverse times output from journey simulation (`outT`). This traverse time is then added to the departure time to give the arrival time at the end of the journey, which allows the punctually of each train journey to be evaluated. Again, the same memory format as `outT` is used for `penT`. All memory access are either coalesced or broadcast via L1 cache.

```
   // for all the journeys made by this train
   while (jID < jID_end)
   {
      tempS = allS[jID];
      // find the departure time
      dwell_time = fmax (tempS.get_minDwell (),
                         allDwells[jID * pSIZE + pID]);
      if (with_noise)
          dwell_time = fmax (dwell_time,
                     tempS.get_meanDwell () +
                             tempS.get_sdDwell () *
                        curand_normal (&local_curandState));
      t = fmax (tempS.get_depart (),
                   float (t + dwell_time));
      depT[jID * pSIZE + pID] = float (t);


      // find the arrival time and calculate the punctuality
      t += outT[jID * pSIZE + pID];
      penT[jID * pSIZE + pID] = (float)t-tempS.get_arrive();
      ++jID;
   }
   if (with_noise)
      RNGStatesSim[tID * pSIZE + pID] = local_curandState;
}
```

Once the true departure times for each journey have been calculated, it is trivial to synchronise signal block events. Efficient memory access patterns can be achieved for this using the same parallelisation strategy as used for the journey simulation stage in Section 5.4.3.

5.4.5 Compatibility check

Once all the timings have been synchronised, signal block events must be compared to establish to what extent each candidate network control strategy results in train movements that that maintain safe operational headways. In the case that no headway violations are found, this gives the same safety guarantees as a signalling system. However, checking the compatibility of signal block events only occurs after all train movements have been simulated. This makes it very different to a signalling system, where train movements are affected by the state of the signals. The design decision to have 'soft' headway constraints was made in Section 5.3.1, and enabled the choice of parallelisation strategy used for journey simulation. Readers familiar with railway signalling may initially be horrified by the idea of 'soft' headway constraints. However, this does not mean unsafe candidate solutions will be tolerated in the final result. Instead, it is a common optimisation technique used to bias candidate control strategies during the optimisation towards better solutions.

Before the level of compatibility between interlocked signalling blocks can be calculated, it is necessary to pair together the entry and exit events that together define how long a train occupied that block for. First the blocks events are sorted by their block index (`bID`), then by the train index (`tID`), and then by the times at which the order events occur. This order turns out to be independent of the control strategy applied so, instead of performing this sort after every round of evaluation, the mapping between the array of signal block events output form journey simulation (`outB`) and the sorted array of

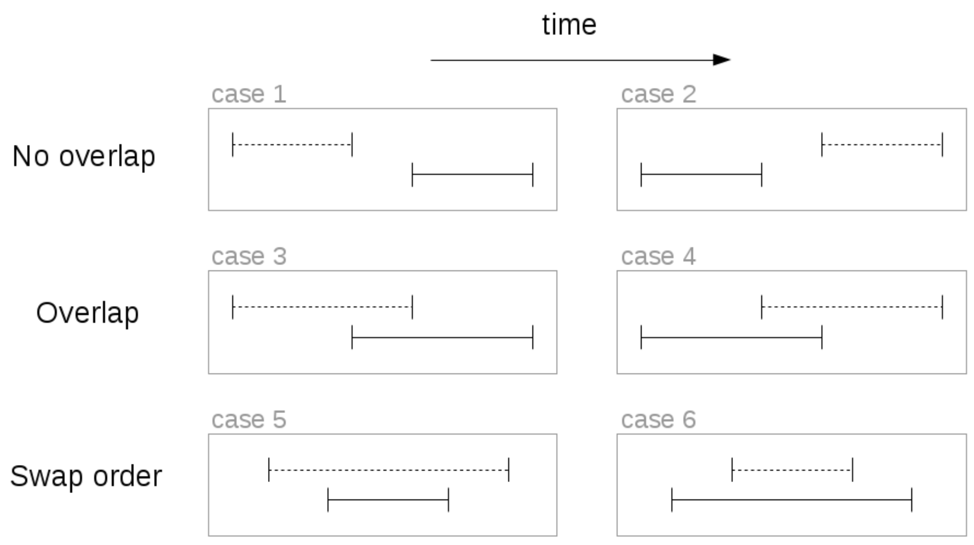


Figure 5.14: The six possible situations when comparing exit and entry times of two signal block occupancies.

block events (`sortB`) is stored in another array. By iterating through the indices of sortB, the sorting and pairing of block events can be done together, minimising the number of memory transfers. If no pair is found then it can be assumed that the train entered the block a time minus infinity, or else exited the block at time plus infinity. Again, good memory access patterns can be ensured by consecutive thread should operate on the signal block events with a consecutive population index `pID`.

Once the sorted array of signal block occupancy pairs has been created, the interlocking between signal blocks is used for compatibility checking. The first stage when checking compatibility between two blocks is to establish if the time period they are occupied for overlap. Figure 5.14 illustrates the six possible situations when comparing timing of two blocks. In cases 1 and 2 there will be no interaction between trains as they the times they are in the signal block do not overlap. As well as specifying which (dependent) signal blocks are effected by each (independent) signal block, the interlocking data must also specify how they are interlocked. This is done be specifying the maximum permissible exit velocity (`v_exit`) as discussed in Section 5.3.4. If two signal blocks interlocked and their occupancy overlaps then safe operation is only guaranteed if the speed of the train in the dependent block is below `v_exit`. If this is not the case then block compatibility checking will give a signal block penalty (`penB`) proportional to the magnitude of the over-speed. A special case is when `v_exit` = 0, which can never be kept as the velocity of all trains will always be greater than `V_MIN` (see Section G). This means `v_exit` = 0 is equivalent to a red aspect being displayed at the entry of the dependent signal block if the independent signal block is occupied. In this instance, cases 3 to 6 are penalised proportional to the minimum difference in time that that would be needed to avoid a collision. For cases 3 and 4 this is simply the time the overlap occurred for. This gives the optimisation important information about which candidate solution is closer to achieving safe operation. Again, good memory access patterns can be ensured by consecutive threads operating on the signal block events with a consecutive population indices (`pID`). Since, at the very least, the occupancies of each signal block must be compared against themselves, one CUDA block was launched for each signal block.

Within each block, `sortB` elements are not ordered by time but first by trains index (`tID`) and then by time. This means that an all on all comparison must be used when checking the compatibility between different block occupancies. While this did not result in excessive execution times when optimising network N1 (see Section 5.7) it will scale as the square of the number of elements in each block. An improvement that could be made would be to add another step, sorting `sortB` by `bID` and then time. This would allow early termination when comparing signal block occupancies.

5.4.6 Scoring

The final stage of evaluating a population of candidate control strategies is to output a score. This score was found by summation of the penalties `penE`, `penT` and `penB`, produced in Sections 5.4.3, 5.4.4 and 5.4.5 respectively. Yet again, good memory access patterns can be ensured by consecutive thread should operate on the signal block events with a consecutive population index `pID`.

5.5 Optimisation algorithms

The GPU accelerated simulation G6, is compatible with any type of population based heuristic optimisation. In order to minimise data transfer between the host and the device it is desirable to also run the optimisation on the GPU. For simplicity of efficient implementation on the GPU, a fine-grained genetic algorithm was implemented, although roulette-wheel selection could also be used (see appendix III) but would require an additional step to sort scores. Once selection has taken place the, using the fine-grained GA described in Section 5.5.1, genetic operators must be applied the to new generation of candidate solutions. Model G6 has different different control variables to previous models:

- each control distance (d_x) now lies within a fixed range
- each control action(c_x) is now a continuous variable (between -1 and +1)
- a new control variable to define the minimum dwell time at each stop is now introduced

This means that the previous genetic operators cannot be used unmodified. Also, since it is important to minimised the amount of global memory access,

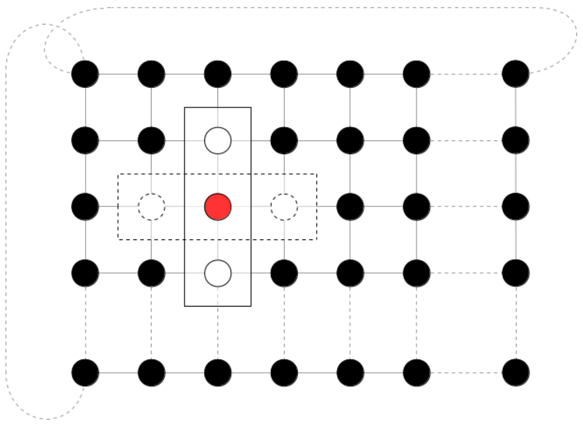


Figure 5.15: The structure of the spacial population for the fine-grained GA. Candidate solution on the edges of the population are neighbours with the opposite end of their row or column. Two parents are selected from the vertical and horizontal neighbourhood (both including the central point).

both mutation and crossover operators are applied in the same stage, described in Section 5.5.2.

5.5.1 Fine-grained selection

Fine-grained GAs map the population to a spacial distribution and perform selection and reproduction locally. This maps well to GPU hardware Yu et al. [81]. The fine-grained GA implemented in G6 maps the population to a 2D network, illustrated in Figure 5.15. Since two parents must be selected, elitist selection was used to select the first parent from the vertical neighbours and the second parent from the horizontal neighbours.

5.5.2 Breeding

Once two parents have been selected, reproduction be carried out. Since the control strategies for each journey can be changes independently of one another on CADA block was launched for each `jID` with consecutive threads operating on control strategies with consecutive population indices `pID`. This is the parallelisation used in journey simulation. First the parent was chosen based on the probability of crossover. This means that whole journeys are exchanged between each candidate solution rather than performing the crossover half way through the journey. The probability of mutation was also applied to each journey. If mutation took place the dwell time and each control point were modified randomly (by predefined standard deviations) whilst copying from the parent to the new population index. By using a fine grained GA, memory throughput is optimised as there is a increased chance that consecutive threads will access control strategies with consecutive population indices `pID`. It may be that the spatial mapping of the population can be optimised to increase the chance of this even further.

5.6 Validation and sensitivity analysis

Sections 5.4 and 5.5 described the development of the new, GPU accelerated, simulation and optimisation G6. It is important to validate the accuracy of this simulation. This ensures that the underlying algorithms are correct and also that they were implementation correctly. However, as G6 uses a distance-stepping simulation (assuming linear acceleration over the small distance interval Δd) the size of Δd used will also affect the accuracy of the simulation. For this reason, validation of G6 is carried out against G5 in combined with a sensitivity analysis of Δd. The validation shows good agreement subject to a realistic choice of problem discreetisation.

Since the simulation used in optimisations G1 to G5 has already been validated against literature (see Section 3.1.4), this simulation was used to validate the accuracy of the G6. To do this, 100 random initialisations were

Table 5.2: Sensitivity analysis of the step length (Δd) using in G6 compared to G5 ($\Delta t = 1$ s). The difference in simulated energy consumption and traverse time was calculated for 600 journeys at each Δd.

	% error in energy		% error in time	
Δd /m	mean	σ	mean	σ
0.01	0.2	0.7	11.8	1.1
0.1	-0.9	0.7	8.1	1.5
1	-1.0	0.8	6.3	1.9
10	-1.1	0.7	1.6	1.8
50	-0.9	0.9	0.2	1.3
100	-0.5	1.1	-0.5	1.2
1000	4.6	10.6	-0.1	112.5

carried out using G5. G5 used the same model parameters as used in Chapter 4, with both training and utilisation noise level set to zero. The best control strategy from each initialisation was saved, along with the simulated energy consumption and traverse time for each journey. These control strategies were of the form described in Figure 3.2, where each control point specifies the distance at which control is switched between maximum traction and coasting. All 100 control strategies were then converted to the form used in G6 (see Section 5.3.5) using a `controlD` of 500 m. This ensured that minimum distance between operational transitions (defined in Table 3.1) could be maintained and all control strategies used G6 stored identical information to those used in G5. These control strategies were then used to initialise G6 (population size = 100) and simulation performed with Δd =1 cm, 10 cm, 1 m, 10 m, 50 m, 100 m and 1 km. The results from this are shown in Table 5.2 and Figure 5.16, and will be discussed starting will large Δd and moving to smaller Δd.

The simulation in G6 uses a distance-stepping model (assuming linear acceleration over the small distance interval Δd) so it is expected that using a large Δd will result in poor approximation of non-linear characteristics. Since

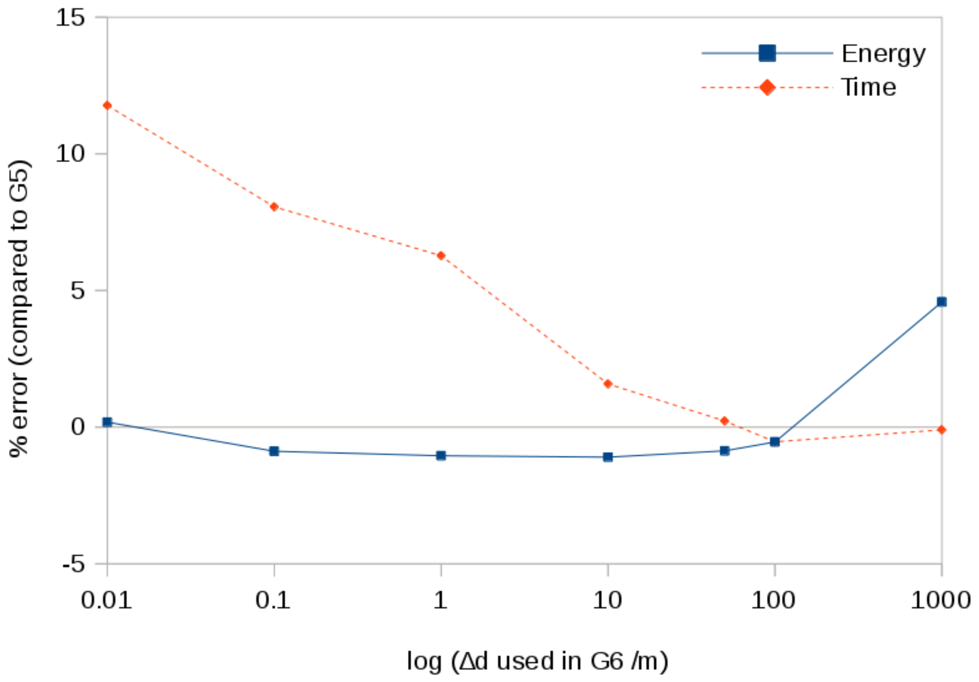


Figure 5.16: Sensitivity analysis

the maximum traction force of the train decreases with velocity it is unsurprising that the simulations using $\Delta d = 1$ km yield trajectories with faster acceleration and therefore an increased average energy consumption on each journey. However, using the same logic, the average traverse times for each journey should be significantly reduced, but this is not the case. The reason for this is alluded to by the large standard deviation in the journey time error (over 112% for $\Delta d = 1$ km). Each control point is only applied from the distance step after it is passed, so increasing Δd causes a decrease in the accuracy with which control is applied. This effect is illustrated by the trajectories in Figure 5.17. In the 100 network control strategies simulated at $\Delta d = 1$ km this led to one particularly extreme case, where a traction control point was superseded by the next coasting control point before it was ever applied. This caused the train to coast until it reached the minimum allowed velocity (0.1

m/s), resulting in an extremely long traverse time of 12690.5 s. Without this one journey the mean error in traverse time would be -1.9%.

It would be possible to modify the simulation in G6 to reduce the extent of errors caused both of these effects at large Δd. Firstly, a Runge-Kutta method could be used to reduce the error caused by assuming constant acceleration over each distance step. Secondly, maintaining a minimum distance between operational transition which is greater than Δd would ensure that no control points are 'lost' during simulation. Also, where control changes, it would be possible to take an additional step in the simulation. This would ensure that each control point is applied at exactly the right distance, but is likely to cause thread divergence (and therefore increased execution time on the GPU) unless $\Delta d \geq$ `controlD`. However, this idea may be worth implementing for journey features, which will sufferer from the same decrease in accuracy at large Δd but will not cause thread divergence. A compromise, to increase the accuracy of control application without causing thread divergence, would be to average the control applied over that distance step. While this is unlikely to give the 'correct' answer, weighting each control according to the distance it is applied for may reduce the total size of the error.

Similar to G6, G5 uses a discretized model but assumes a linear acceleration over the small time interval $\Delta t = 1$ s. It is likely that the two models will best agree when resolutions of the simulations are most similar. The mean traverse time simulated by G5 was 772.4 s which, given $\Delta t = 1$ s and journeys of 30 km, is equivalent to an average distance step of 38.8 m. This fits well with the observations in Figure 5.16, where the simulation with G6 and $\Delta d = 100$ m gave the smallest difference in traverse time and energy consumption compared to simulation by G5. $\Delta d = 10$ m was also relatively similar but with a slightly decreased average energy consumption and increased traverse time over each

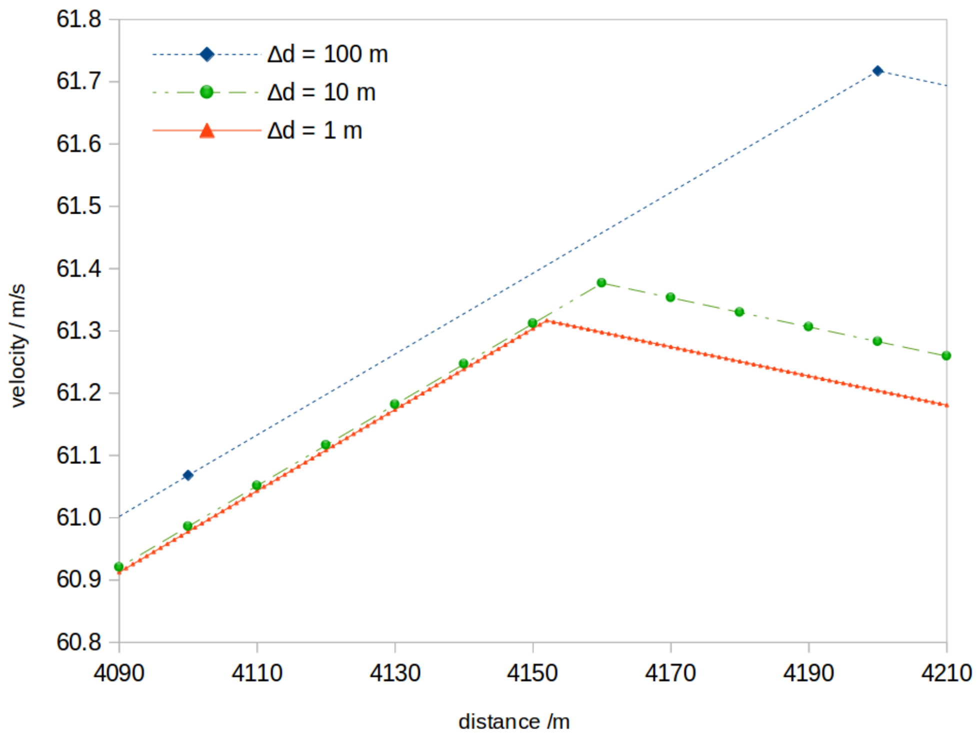


Figure 5.17: Train trajectories simulated by G6 illustrate two sources of modelling error caused by taking large Δd steps. (i) before reaching the first control point the train velocities have diverged due to reduced accuracy with which large Δd can model the non-linear traction characteristics of the train. (ii) the first control point is at 4151.9 m but is only applied from the distance step after it is passed. This leads to a reduction in the accuracy of control point application as Δd increases, in this case longer application of maximum traction before coasting is applied.

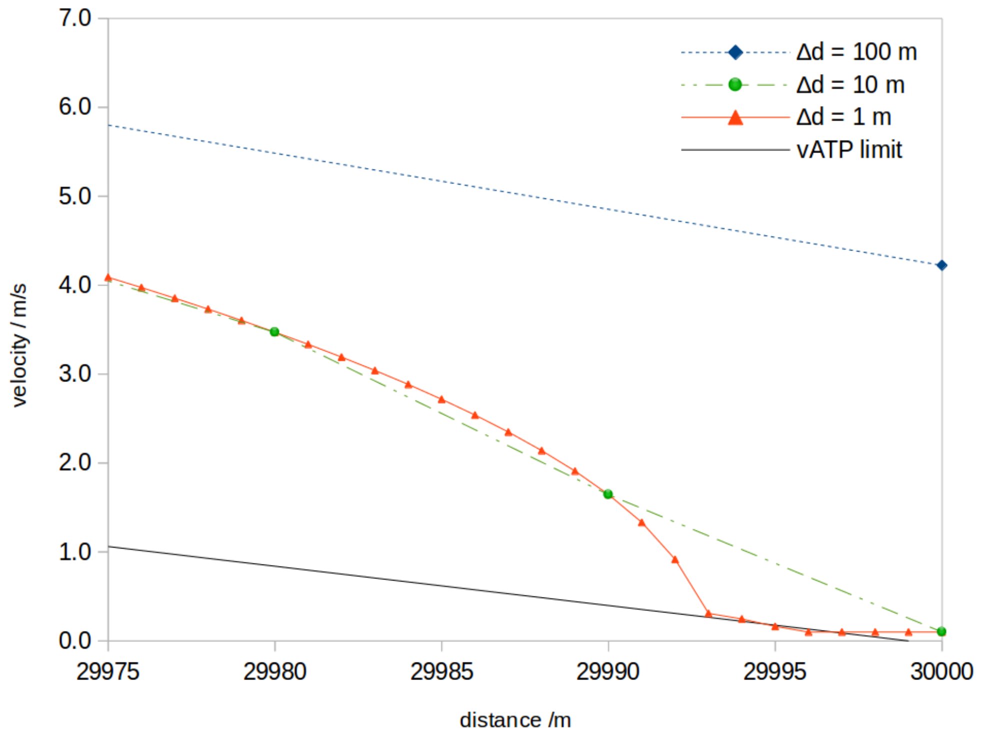


Figure 5.18: Train trajectories simulated by G6 - brake to stop.

journey. This difference between G5 and G6, when $\Delta d = 10$ m, is likely due to the increased accuracy of G6 resulting in simulation of train trajectories with slightly reduced acceleration. This effect is illustrated in Figure 5.17 and accounts for the difference in velocity when passing the first control point at 4151.9 m.

As Δd decrease below 10 m it might be expected that the simulation would be able to model non-linearities even more accurately. However, while this appears to be the case at $\Delta d = 1$ m, the increase in accuracy over $\Delta d = 10$ m is small and other factors lead to a decrease in the overall accuracy of the simulation. Figure 5.18 illustrates the largest of these factors - how the train brakes to a stop at the end of a journey. Braking is triggered when a train's velocity is greater than the automatic train protection speed limit (vATP) defined by a series of journey features (see Section 5.3.3). In G6 the vATP limit is re-calculated each distance step using single-precision floating point arithmetic. Floating points numbers have a limited precision so lose accuracy when adding numbers of very different scales or subtracting very similar numbers. In this case, vATP is very close to zero so loss of precision is caused by the later and will increase if longer journeys are simulated. When simulating with $\Delta d = 10$ m the rounding error in vATP is not a problem as it can be seen in Figure 5.18 that the train applies maximum service braking and is almost stationary at the end of the journey. It is expected that the simulated trajectories will be above the vATP limit as this limit was calculated by making a piece-wise linear approximation, rounding down from the maximum braking trajectory (see Section 5.3.3). However, the trajectory resulting from the $\Delta d = 1$ m simulation using G6 is able to reach the vATP limit before the end of the journey causing it to travel the last 4 m of the journey at the minimum allowed speed of 0.1 m/s. This final 4 m will add just under 40 s to the overall journey time. If all journey times in the $\Delta d = 1$ m simulations were extended by a similar amount then this would account for the 4.7% increase in the average journey time between $\Delta d = 10$ m and $\Delta d = 1$ m simulations (see Table 5.2).

To overcome this, the vATP limit could be defined using:

$$\text{vATP} = m(d + c\prime) \tag{5.12}$$

where m is the same as the constant m used in Equation (5.1) and $c\prime$ is a related to the constant c in Equation (5.1) by the relation $c\prime = c/m$.

This allows the subtraction to occur before the multiplication so, since Section 5.4.3.A has already considered the accuracy associated with a single-precision floating point representation of distance, we can be confident that Equation (5.12) will not suffer from the same rounding error as Equation (5.1). Also, since the train can not come to rest during the journey simulation, it makes sense to specify an allowed level of residual velocity at the end of the journey. This could take the form of low but constant speed limit at the end of the journey, the exact form of which is a topic for further research.

Finally, at very small Δd (≤ 1 cm) it is likely that accumulation of floating point arithmetic errors further decreases accuracy of the simulation. The use of double-precision floating point numbers is probably the best way to reduce this sort of error, but will come at the cost of a greatly increased execution time on most consumer GPUs.

5.7 Measuring performance

Having established in Section 5.6 that the accuracy of G6 shows good agreement for $10 \leq \Delta d \leq 100$ m, the execution speed of G6 can now be investigated. This was carried out using a relatively low specification desktop computer, 2.4 GHz Intel Core Duo E4600 with 2 GB RAM, equipped with one 1.084 GHz GeForce GTX 750 Ti with 2 GB RAM. To allow comparison with previous models, optimisations using G6 were performed on network N1, using the same target schedule and cost function as Chapter 4. Since $\Delta d = 50$ m has been shown to give a similar accuracy to previous models, this was chosen as the step length for simulation. Also, the population size used was varied to demonstrate the full potential of G6. Execution times included the time spend reading in the problem description and the random initialisation of the population, and were recorded using the time elapsed between the start and end of each optimisation. The results of these optimisations are shown in Table 5.3.

Table 5.3: Comparison of the rate at which different models can evaluate candidate solutions. Y1 and G1 both used $\Delta t = 1$ s and their execution times were from [44, Table 4] and Table 3.3 respectively. Optimisations using G6 used $\Delta d = 50$ m and were repeated 10 times.

model	`pop_size`	Generations	mean execution time: whole optimisation /s	mean execution time: per candidate solution simulated /ms	speed-up factor (per candidate solution simulated)
Y1	40	800	463	14.5	1
G1	40	800	82.4	2.58	6
G6	40	800	0.658	0.0206	704
G6	64	800	0.663	0.0130	1117
G6	1024	800	1.54	0.001889	7673

It is clear from the results in Table 5.3 that G6 affords a significant speed-up compared to both Y1 and G1. The same simulation used in G1 was also used in G2 to G5. When `pop_size` = 40, G6 simulated each candidate solution over 100 times faster than G1. However, it is expected that the efficiency of computation will increase if `pop_size` is a multiple of 32. This is because GPUs execute threads in batches of 32, and the parallelisation strategy (discussed in Section 5.3.2) means the total number of threads is a multiple of the population size. When `pop_size` = 64, the optimisation evaluated 60% more candidate solutions compared with `pop_size` = 40, but the total increase in simulation time was less than 1%.

Also, as discussed in Section 5.2.2, GPUs hide memory latency by switching between different blocks of threads. This means enough blocks must be active to hide memory latency and minimise the total execution time. For the journey simulation stage, the number of CUDA blocks is determined by:

$$\text{number of journeys} * ceil(\frac{\texttt{pop\_size}}{\texttt{TILE\_SIZE\_pop}}) \tag{5.13}$$

where the *ceil* operation rounds up to the nearest integer. Only 6 journeys are being simulated in this optimisation and `TILE_SIZE_pop` = 128, so `pop_size` = 64 will only allow CUDA to launch 6 blocks. This will not be enough to all to fully hide hide memory latancy on the GTX 750 Ti used in this investigation. It is possible to increase the number of CUDA blocks available for the GPU to schedule by increasing the `pop_size`. Using G6 with `pop_size` = 1024 means a total of 48 blocks are launched for the journey simulation stage. Again, the results in Table 5.3 show that increasing `pop_size` greatly increases the aggregate speed with which candidate solutions are evaluated. This is due the increased latency hiding when launching more blocks and allows G6 an increase in execution speed of three orders of magnitude compared with G1. For `pop_size` = 1024, the journey simulation stage (discussed in Section 5.4.3) took 92% of the total kernel execution time, suggesting it is still the most critical step determining the total execution speed.

While the three orders of magnitude speed-up between G1 and G6 is useful in practice, it is specific to both the algorithm implementation and the hardware used to execute them. We can get an idea of the potential speed-up due to hardware by comparing the theoretical throughput of the specific hardware used. The CPU used to evaluate G1 was an AMD Phenom II N850@2.2 GHz (see Section 3.1.5). No multi-threading was implemented and double precision arithmetic was used so the theoretic throughput of 8.8 GFLOP/s is given by: the number of double precision floating-point operations per second (4 for the AMD K10 processor family) multiplied by the number for cycles per second (the clock speed of 2.2 GHz). G6 was evaluated on a GTX 750Ti GPU which is listed as having a theoretical peak throughput of 1,389 GFLOP/s for single precision floating-point operations. This suggests that of the reported 1238 times speed-up factor between G1 and G6, a factor of over 150 is potentially due to hardware differences alone, with the other factor of 8 split between algorithmic improvements and a more highly optimised implementation for the GPU architecture. However, this speed up due to hardware is only made possible because the new algorithms have been optimised for the GPU architecture. Many of the same optimisations developed for G6 (e.g. parallel execution, safe use of single precision floating point variables, journey features, etc.) should be equally effective in optimising CPU based algorithms. Papers such as Lee et al. [82] are a reminder that it is important to optimise the CPU implementation before a fair comparison can be made. However, due to the trend in Figure 5.3 it is expected that the benefit from execution on GPU architecture will still afford a 10 times speed-up, even with similar levels of optimisation for the CPU and GPU implementations.

G6 has shown to greatly increase the speed at which candidate solutions can be evaluated compared to previous multi-train simulations. When performing heuristic optimisations the ability to evaluate the performance of candidate solutions faster is always advantageous. This could be used to give the same level of optimisation in a shorter time, better local optimisation (by running the optimisation for more iterations), or a better ability to cope uncertainty and avoid local minima (by using a larger population size). However, due to the new form of control strategy implemented in G6, effective optimisation will only be achieved once the optimisation algorithms and parameters have been tuned this specific representation of the multi-train trajectory planning problem. This should not be considered a trivial task and is analogous to the work described in Chapter 3. To illustrate this, the trajectories resulting from 10 optimisation with G6 (`pop_size` = 40) are shown in Figure 5.19. This is similar to Figure 3.11, illustrating both a strong consensus between the solutions in some areas and a large degree of local variation in other areas.

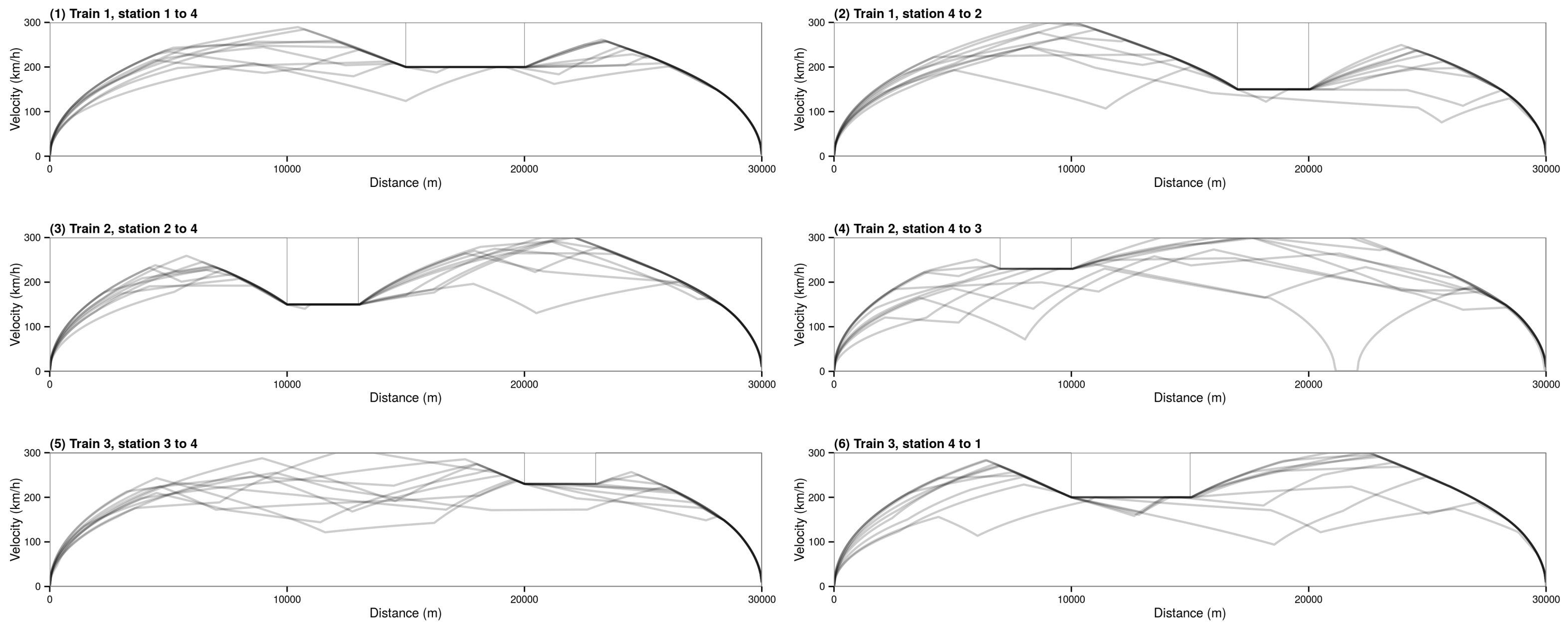


Figure 5.19: The consistency of train trajectories found using G6 (`pop_size` = 40) to optimise N1. Trajectories from 10 independently optimised network control strategies are overlaid.

Chapter 6

Conclusions and further work

6.1 Conclusions

In this thesis, several new algorithms have been proposed and demonstrated to improve multi-train trajectory planning. A generalizable procedure has been proposed for multi-train trajectory planning in the presence of uncertainty and a new multi-train simulator has been developed for accelerating population based heuristic optimisations of train movement.

In Chapter 3, several improvements were proposed and demonstrated to advance the capability of the multi-train trajectory optimisation originally proposed by Yang et al. [44]. The published model and optimisation was implemented as G1 and validated against the published results. After carrying out repeated optimisation using G1, the control strategies produced were examined and two problems identified. Firstly, there was a large local variation in the position of some control points after optimisation, suggesting poor local optimisation. A less-constraining mutation operation was proposed in Section 3.2 and demonstrated to improve the local optimisation of the GA. Secondly, once the local optimisation performance had been improved, a specific type of local minimal was also identified in Section 3.3. For a trajectory defined by traction coasting pairs, the mutation operation introduces variation in the position of the control points. If the control strategy becomes limited by a region of reduced line speed, and this region is much longer that the maximum mutation distance, then unfavourable intermediate control strategies must persist for several generations in order for a control point to pass from one side of the region to the other. To overcome this, and ensure that control points cannot become excluded from a region where they are needed, a procedure to insert and delete pairs of control points was proposed. These were shown to further increase the quality of solutions found by the GA optimisation. These improved operations were combined in optimisation G4, the performance of which was investigated in Section 3.4. G4 was shown to optimise an average of 27.6% further than G1 when compared with randomly initialised solutions. This was achieved in combination with increased consistency (1/28th of the standard deviation in objective score of solutions), and faster GA convergence (less than one-quarter the number of generations). This improved optimisation consistency allowed a more detailed investigation of the effect of varying α, the weighting between different objectives in the cost function, to be conducted in Section 3.5.1. For the system studied, the components of the objective function respond like step functions with regard to variation in α, causing the optimal objective solutions to switch rapidly between the extreme solutions of minimum time and minimum energy.

When planning train trajectories it is important to consider the robustness of control strategies if they are to be implemented in real systems. Real systems contain many uncertainties, two of which were investigated in Chapter 4: the accuracy of control point application, and variations in station dwell times. The highly optimised control strategies found by G4 performed well when utilised in a system with no uncertainty, but quickly degraded as the level of uncertainties increased. To address this, a new, genetic algorithm based, optimisation procedure was described in Section 4.2. This procedure seeks to find robust solutions to the multi-train trajectory planning problem and is easily generalizable to include many different uncertainties in the system. Here it was implemented as optimisation G5, and shown to be effective in finding robust control strategies in the presence of both types of uncertainty under investigation. These uncertainties were first considered separately, in Sections 4.3.2 and 4.3.3, before it was demonstrated, in Section 4.3.4, that they could be considered simultaneously in the optimisation and still achieve similar levels of robustness. For both types of uncertainty investigated, a trade-off between the robustness and the expected score of the solution was observed, reminding us that robustness is not cost free. This means that for best results the training noise level used during the optimisation progress should reflect, as accurately as possible, the noise level that will be experienced when the optimised control strategy is utilised. A procedure for estimating the performance of a closed-loop optimisation was also developed and investigated in Section 4.4. As would be expected, this achieved better performance than open-loop solutions found by both the non-robust (G4) or robust (G5) optimisations. However, for the system and noise levels investigated, the robust open-loop solutions were found to afford up to 55% of the benefit of closed-loop control (compared to non-robust solutions). This suggests the proposed robust optimisation may be worth further investigation, especially considering that open-loop solutions can influence implementation (e.g. via DAS) without the communication infrastructure and real-time optimisation required by for optimised closed-loop control.

Chapter 5 documents the development of a new multi-train simulator, designed to accelerate population based heuristic optimisations using a GPU. This new model and optimisation, G6, also removes many of the limitations of the model used in optimisations G1 to G5. Firstly, soft constraints mean information stored in invalid candidate solutions is no longer discarded, reducing the number of wasted calculations. They also allow a high level of parallelisation, which is important for algorithms targeting a GPU architecture, and make integrated scheduling and trajectory optimisation possible to some extent. Secondly, the simulation is now capable of modelling junctions, allowing more realistic networks to be modelled. Thirdly, as well as encoding the position at which it should be applied, each control point also encodes the full range of control action possible (from maximum service braking to maximum traction). This increases the number of control strategies that can be defined. Fourthly, G6 was designed for efficient implementation on parallel architectures. All parts of the simulation and GA were moved onto the GPU, removing the need to slow memory transfers between the host and device. Also, the data structure and algorithms proposed were designed together, to maximise coalesced memory access on the GPU. Finally, in Section 5.6, G6 was validated against previous multi-train simulation. A sensitivity analysis of Δd, the resolution used during journey simulation, was conducted and showed agreement subject to realistic choice of problem discreetisation. For $10 \leq \Delta d \leq 100$ m the average energy consumption and traverse time of journeys simulated by G6 were within 1.6% of the values evaluated by G4. On a low specification desktop computer, equipped with a £110 GPU, G6 was able to simulate journeys at an aggregate rate of over 95,000,000 km/s ($\Delta d = 50$ m). This constitutes a 3 orders of magnitude speed-up over G4 ($\Delta t = 1$ s) and is the equivalent of simulating a 250 km journey (e.g. from Sheffield to London) in under 2.5 μs. When performing heuristic optimisations the ability to evaluate the performance of candidate solutions faster is always advantageous. However, due to the new form of control strategy implemented in G6, effective optimisation will only be achieved once the optimisation algorithms and parameters have been tuned this specific representation of the multi-train trajectory planning problem. This should not be considered a trivial task and is analogous to the work described in Chapter 3. Once an effective optimisation has been developed to use with G6, the greatly increased rate of simulation could be used to give the same level of optimisation in a shorter time, better local optimisation (by running the optimisation for more iterations), or a better ability to cope uncertainty and avoid local minima (by using a larger population size). These would intern increase the feasibility of investigating a larger number of other application of multi-train trajectory planning, some of which are discussed in 6.2.2.

6.2 Further work

The investigations undertaken have raised many research questions beyond the scope of this work. In particular, the GPU accelerated model described in Chapter 5, G6, has many potential applications. However, due to the new form of control strategy implemented in G6, the optimisation parameters must be optimised and new genetic operators developed before optimisation using a GA will be effective. This is not a trivial task and is analogous to the work described in Chapter 3. As well at the more theoretical improvements (outlined in Sections 6.2.1 to 6.2.4) G6 would also benefit from some practical improvements. Improved pre-processing of input data, such as converting line speeds to `FEATURE_TYPE_vATP` journey features, would increase the usability of the optimisation and therefore the number and complexity of applications that could be investigated. Better visualisation tools would also aid in the analysis of results. Also, compatibility with standardised data formats, such as RailML [83], would be highly desirable. RailML was being developed concurrently with the research reported here so was not available at the beginning of the project.

6.2.1 Sensitivity analyses

A sensitivity analysis of Δd has already been conducted in Section 5.6. However, before considering potential applications of the current model, its sensitivity to other variables should be carefully investigated.

Enabled by faster computation and communication technologies, many capacity constrained railway networks are moving towards the use of moving-block signalling systems. G6 cannot model these directly, but can approximate them with arbitrary accuracy by using journey features to add more signalling blocks (with more complex interlocking). This concept is described in Section 5.3.4. A sensitivity analysis should be conducted into the number of signalling blocks used to approximate moving block signalling. Also, the size of the distance step, Δd, during simulation will affect the accuracy with which signal block events can be recorded. This means the effect of Δd size should also be considered with respect to approximating moving block signalling. As well as checking the accuracy of results, it would be interesting to know how the performance of compatibility checking scales with the number of signal blocks being modelled. Since it was not limiting in the case studies investigated, the algorithm used to check compatibility (interlocking between signal block occupancies) has not been carefully optimised. This issue of scalability could also be investigated more generally. For example, how do the total number of trains, journeys, journey features, interlocking, control points, etc. affect optimisation performance? This could be split into two categories: (i) how different algorithms involved in simulation affect the total computation time of each generation (ii) how different systems (problem spaces) affect the rate of GA convergence.

Another sensitivity analysis that could be conducted is the effect of different sources of uncertainty on the robust optimisation methodology described in Chapter 4. However, apart from station dwell times, literature on the uncertainties associated with different parameters seems to be quite sparse. This may be because this information has traditionally been difficult to measure, as reliable uncertainty measurements can require a lot of data collection. However, the increased used of sensors monitoring many components in the railway industry is leading to the production of large quantities of data. This 'big data' may well contain sufficient detail and repeat-coverage to, not only estimate parameters needed for modelling, but also estimate the uncertainties associated with them. Whether the actual uncertainties can be obtained or not, it will still be possible to investigate the magnitude of different uncertainties needed to significantly affect the outcome of the robust optimisation. These uncertainties can also be combined during the optimisation, although if the total uncertainty becomes too large then the optimisation will not converge. Increasing the population size will help the optimisation function effectively with higher levels of uncertainty, so it may be that other modifications to the optimisation will also improve its performance. Further modifications to the optimisation algorithm are discussed in Section 6.2.4. However, in situations where the optimisation does not converge it may be that that system is not suitable for (off-line) trajectory planning.

6.2.2 Potential applications

The model (G6) developed in Chapter 5 showed greatly increase simulation speed which opens up many potential applications.

The current multi-train trajectory optimisation could be used to investigate the effect of degraded operation of sections of track, on the overall network performance. If this degraded operation takes the form of a reduced line speed then it can easily be incorporated into the model. Comparing the performance of optimised train movements, before and after the line speed is reduced at a particular location, will indicate the impact of of that change on the system. If this process was repeated over a whole network then a 'heat map' of that network could be created indicating the areas where it is particularly important to avoid degraded operation. This could be helpful when targeting maintenance work or considering where it may be worthwhile investing in higher quality infrastructure components. It is expected that the results from this would simply show with the busiest sections of track identified as the most critical. However, it is also possible that multiple infrastructure components may fail at the same time, causing degraded operation at two locations. Network interactions may mean these have a reinforcing effect, where two failures have a greater impact than the sum of the failures separately. Identifying these second order interactions is a much larger problem due to its combinatorial nature. It may be that G6 is fast enough to make solving this problem possible on some realistic networks. Conversely, the same method could be used to examine the benefit of upgrading areas of track to allow an increase in line speed.

Another problem that could be addressed is how to develop schedules that are robust to small perturbations. The current optimisation has no facility for re-routing of trains so small perturbations are those which do not require re-routing or cancellation of services. The closed-loop optimisation technique described in Section 4.4 can then be used to assess the impact of a particular perturbation instance. A Monte-Carlo method can be used to estimate the impact of likely disruptions allowing the robustness of a schedule to be quantified. This information can then be used as a fitness function allowing the robustness of the schedule to be optimised, though this would require a schedule optimisation algorithm to be developed.

In fact, there are certain situations in which G6 can already perform some implicit schedule optimisation. The last stop must have a scheduled target time otherwise there will be no incentive for the train being optimised to go fast and the resulting trajectory will not be realistic. However, if no target arrival or departure times are scheduled for intermediate stops the optimisation will choose the times which allow it to best meet its other objectives. This would be an example of a "truly integrated scheduler and train control optimizer" of the sort McClanachan and Cole [9] could not find in literature. Also, this situation could equally apply to routing trains through a junction. The current schedule does not explicitly define the order trains must pass through a junction, though where trains have a common route before or after the junction it may define this implicitly (using the scheduled arrival or departure times). In either case, the trajectories found by the optimisation will determine the actual order the trains pass through the junction.

It would also be interesting to see if the trajectories found, using the optimisations described in this thesis, could be utilised in a real system. However, since different systems use different methods of control, the specifics of how this might occur would vary. It is likely that many ATC systems use proprietary software making target trajectories difficult or expensive to modify. Also, since a change will constitute modification of a safety critical process, it is likely to need extensive testing and safety validation. However, the majority of trains on the GB mainline are still controlled by drivers. In this case, the drivers ensure safe operation but their behaviour may be more easily modified. Driver Advice Systems (DAS) exploit this and try to improve the performance of a system, either by giving the driver direct advice on how to drive (e.g. target speeds) or by giving them additional information (e.g. time ahead or behind the schedule). This may be a more feasible route to testing the success of optimised trajectories. It is thought that the robust optimisation described in Chapter 4 may be particularly useful in this type of application, as it can take into account the level of accuracy with which a driver may apply the control strategy. Also, a related application, would be to compare the performance of trajectories from current driver against optimised trajectories. This may allow improved operational performance through allowing more tailored driver training but would also have the potential to work the other way, highlighting areas where the optimisation or simulation could be improved.

As well as training real drivers, the optimised trajectories could be used to optimise the behaviour of simulated driver algorithms. To some extent drivers allow a railway network to be modelled as a multi-agent system - each driver makes independent decisions based on limited information on their local surroundings. Many networks will be so large that real-time multi-train trajectory optimisation of the whole system at once is likely to remain infeasible. However, agent based control would allow some optimisation to take place at a local level. In optimisations that use ATO to enforce headway constraints (for example [39]) the driver behaviour is an important factor in determining the overall network performance. By tuning agent behaviour (i.e. driving styles) it may also be possible to optimise the performance of the network as a whole. This raises research questions such as, how much information does each agent (driver) need in order to allow them to make good decisions? One approach to optimising agent behaviour would be to use many examples of optimised networks to train an Artificial Neural Network (ANN). The performance of the ANN driver could then be assessed by using it to control trains in a conventional multi-train simulator such as BRaVE[84].

Another approach to optimising the performance over a very large network would be to split it into a number of more manageable sub-problems. These sub-problems could then be solved using the multi-train optimisations described in this thesis. This is linked to the idea of distributed control where, instead of one central controller controlling the whole system, a number of separate control elements are distributed throughout the system. The GB mainline railway network already operates using distributed routing and signal control. Traditionally this has been done at a very local level using thousands of signal boxes, but this has gradually been consolidated until soon only 12 Rail Operating Centres (ROCs) will control the whole network[85, 86]. It is likely that the current optimisations would work at the relatively small (signal-box) scale. If each local optimisation could be performed independently then this would provide a scalable approach to optimising larger networks. However, this requires predefined boundary conditions (i.e. train locations, speeds and times) at the start and end of each optimisation. In reality, unless boundaries are at scheduled stops then it will be difficult to predefine the boundary without restricting the optimisation. How to handle these boundaries efficiently would be a topic for further research. It is possible that the local optimisation could be carried out in parallel and the boundary conditions updated between each iteration. This would lend itself well to scaling G6 across multiple GPU, since the boundary conditions are a very small amount of data and could be transferred asynchronously to avoid blocking execution.

6.2.3 Extension of the model

As well as the applications currently possible there are also a number of applications that would become possible with relatively minor extension of G6.

Once these more fundamental investigations have been performed on the current implementation of G6, it may be desirable to extend the simulation so that other factors can be considered in the optimisation cost function. Including power networks in the simulation is an obvious next step, especially as accurate simulation of energy consumption is such a critical aspect of trajectory optimisation. There has been a reasonable amount of research into this already (see Section 2.2.2) but time constraints meant it could not be implemented during the course of this PhD. The concept of journey features (introduced in Section 5.3.3) was designed to allow easy incorporation of distance based track features within the model. For example, two new types of journey feature `FEATURE_circuit_entry` and `FEATURE_circuit_exit` could be used to trigger the recording of the time and Δenergy at the boundaries between different power distribution circuits. Like signalling blocks, if a higher resolution of data was required for the power network model then more circuit features could be added. Modelling the power network then opens up many more potential applications for the optimisation. These include: reducing power peaks, maximising the use of energy from regenerative braking (particularly relevant on DC networks), designing the layout of new power networks, and investigating the effect of line side (or on board) energy storage. Some of this is being investigated in the EPSRC project TransEnergy[87].

Another, candidate for extending the simulation is to include the movement of passengers. Trajectory optimisation can affect the arrival and departure times of trains and these arrival and departure times are already stored within the model. One cost function parameter that could be calculated using this data is the passenger over-crowding at stations. For example, consider the situation where two trains arrive at exactly the same time at a terminus station. All the passengers on both train will enter the station at the same time, potentially leading to over-crowding and its associated dangers. However, if these trains were to arrive a few minutes apart then it may be that this temporal delay would lead to a more even utilisation of the station's resources. If some measure of over-crowding was calculated then it could be incorporated in the cost function causing the optimisation to try to minimise this as well. Passenger satisfaction is a related area which could be incorporated into the model. There are many factors that affect this but a few that are particularly relevant to trajectory optimisation are: connections, unexpected stops and jerk.

- In a similar way to over-crowding, the relative arrival and departure times of different trains at the same station will effect what connections can be made. Passengers generally want shorter journey times so as well as the minimising the traverse time of individual train journeys, what connections can be made may have a significant effect on the total journey time.

- Passengers on a train may not notice the difference between 50 mph and 100 mph, but they will certainly notice the difference between 50 mph and 0 mph. Unplanned stops between stations are likely to have an adverse impact on the journey quality perceived by passengers. It may be that if the train has maintained a lower speed in an earlier part of that journey it would have arrived at the same time (using less energy) and also avoided having to slow to a speed that annoys passengers.

- Passenger satisfaction may also be adversely affected by excessively high or frequent acceleration and jerk. These are directly due to the velocity profile of the train but are not currently considered in optimisation G6.

For simple problems the new GPU accelerated simulation may allow multi-train trajectory optimisation to take place fast enough to be integrated into automatic control systems. This would allow optimised recovery from minor disruptions. One fundamental issue that must be addressed when implementing this is that a heuristic optimisation using G6 is not guaranteed to find any usable solutions (unless run for an infinite amount of time). For simple problems this does not seem to be a common issue, but if no solution was found it is likely to cause the control systems problems. A simple solution to this would be use to the heuristic optimisation in combination with a simpler optimisation that guarantees finding a solution, even if this solution is not well optimised. The main thing limiting the application of G6 to real-time control is the speed that the multi-train trajectory optimisation can occur. To a large extent this will be determined by the specific optimisation algorithm used. Possible improvements to this are suggested in Section 6.2.4

6.2.4 Improvements to optimisation algorithm

Since GAs were used in this thesis this section suggests a number of modifications to the GA that may improve optimisation performance. However, it should be emphasised that GAs are not necessarily better suited than other types of optimisation for solving the multi-train trajectory optimisation problem. All heuristic optimisations are similar in that they seek to explore a search space, and find the global minima (or maxima) using the fewest possible number of evaluations. To do this they all use information from previous evaluations to concentrate their search in promising areas. As highlighted by Chapter 3, the exact operations performed by the optimisation algorithms have a huge effect on the quality of the resulting optimisation. In fact, the suitability of the operations used is likely to have a larger effect than the ‘type’ of optimisation used. This section focusses on how particular optimisation issues could be addressed using a GA. However, since G6 is compatible with any type of population based heuristic optimisation, future efforts to improve the optimisation could equally well focus on addressing the same issues using other types of optimisation. The two main issues are:

- Focusing effort in promising areas of the search space
- Avoiding local minima in the search space

As discussed during validation in Section 5.6, the simulation accuracy of G6 can be modified by varying the simulation distance step, Δd. It may be beneficial to vary the size of Δd steps during the course of an optimisation. The candidate solutions in early generations are unlikely to be of a high quality and are also likely to be easily distinguishable from each other. This means that simulation using a large step size (with lower accuracy) may yield sufficient information for selection to be effective. Conversely, as optimisation progresses simulation of solutions may need to be more accurate in order to select the best candidate from the population. This may make better use of computational resources, using more accurate (and therefore time consuming) simulation only when it is necessary. Another variation of this idea would be to use a new type of journey feature to modify Δd as a journey is simulated. This would allow a higher simulation resolution in areas where a accuracy is required (e.g. at the start and end of the journey, where there is a change in speed limit, over particularly steep gradients, or around junctions). However, in the intermediate areas, where the train is likely to maintain a steady speed and power consumption, larger Δd may provide sufficient accuracy.

A similar idea is to increase the complexity of control as the optimisation process progresses. Some trajectory optimisations use simple control strategies such as a single target velocity for each train journey [39]. This greatly decreases the size of the search space compared with G6, where both the position and traction setting must be optimised for each control point. A smaller search space will allow promising areas to be identified faster but in doing do is also unlikely to describe true optimal solution. Firstly, one way to apply this would be to limit the number of values control could take. For single train trajectory optimisation it has be shown, see Section 2.1.1, that four operational modes are sufficient to describe optimal control: maximum traction, cruising, coasting, and maximum braking. It is likely that restricting control to these discrete modes would lead to faster convergence. However, in certain situation these may not be sufficient to describe the optimal trajectories of multiple trains, with different tractions parameters. Secondly, increasing the resolution of control as the optimisation progresses may improve the speed with which the search space is narrowed (early in the optimisation). However, it would still allow a finer resolution of control which may be necessary to describe highly optimised solutions. In G6, a simple way to implement this would be to double the control resolution every n generations. If each newly inserted control point was initially set the same as the one preceding it then the phenotype of the solution would not be effected by insertion. Subsequent local searching would then modify these control points allowing optimisation of their positions and traction levels.

Generally, early on in the optimisation the main task is exploring the whole search space and identifying promising regions, whereas late in the optimisation the priority shifts to local optimisation of these areas. It is likely that different optimisations will perform best at these two tasks so ideally this should be reflected by changing the behaviour of the GA. GAs usually try to avoid local minima by allowing some less fit solutions to enter the next generation. In contrast, Greedy algorithms choose the locally optimal solution at each stage, so are specialised at local optimisation. Most GAs, including the ones developed in this thesis, have a parameter defining the selection pressure. At one extreme (selection pressure = 0) the GA will perform a random walk and randomly select solutions to be kept in the next generation. At the other extreme (selection pressure = 1) the GA will turn into a greedy algorithm, exclusively selecting the best solution to populate the next generation. Normally an intermediate value is chosen for the GA but, as discussed in Section 4.2.3, it may be better to vary (in this case increase) this value throughout the progress of the optimisation to cause a smooth transition between the GA and Greedy algorithms. Again, this emphasised the importance of tuning the GA parameters. Ideally, systematic optimisation of all GA parameters should be undertaken.

Another way of focusing effort in promising areas of the search space is to exploit any prior knowledge we may have about the search space. A good example of this is [39] where the allowed range for each target speed is restricted based on the scheduled train service interval and the minimum line headway between trains. Alternatively, if the GA is being used as part of a feedback loop then it is likely that in most cases the new global minima will lie close to the old global minima. This means that the area around the old solution is likely to be a good place to look for the next solution. Both of these would be implemented through GA initialisation. The effect of GA initialisation on overall optimisation performance is another area that could be explored in more detail. This could answer research questions such as: Do the initialisations used cover the search space evenly? If not, what sort of bias do they introduce, and will this help or hinder the optimisation? An alternative method to enhance initialisation for real-time control is to seed the population with a number of different pre-optimised solutions. For example, closed-loop control is likely to have to recover from small perturbations. The closed-loop optimisation technique described in Section 4.4 could be used to optimise a number of probable perturbation instances. Although this might be a time consuming process it could be performed off-line - before the time critical optimisation process has begun. These pre-optimised solutions could then be used to initialise the population. When each real-time optimisation takes place it is likely the situation being optimised will be similar to some of the pre-optimised solutions. This should lead to an increased speed of GA convergence, potentially allowing real-time control in situations where it would not have been feasible with random initialisation. Other techniques for applying evolutionary optimisations in dynamic environments can be found in [88].

As well as speeding up the search it is important to avoid converging on local minima in the search space. GAs try to ensure this by having a population of candidate solutions. However, this relies on there being sufficient variation within the population - if all the candidate solutions are very close together then they could get stuck in the same local minima. There are a number of methods to encourage diversity within GA populations which could be explored. The fitness function could include an explicit penalty for solutions that lie close together. Wei et al. [32] use Hamming distances to quantify the difference between solutions. Hamming distance is the minimum number of substitutions to change one control into another. This works well for discrete control (as used in [32]) but measures such as the square of the Euclidean distance may be more suitable for the continuous control variables used in G6. Also, since both the distance and control action at each control point have predefined ranges, it would be trivial to normalise the comparison metric. The largest difference between each control point distance would be `control_d` and the largest distance between control point actions would be $|c_{max\_traction} - c_{max\_brake}| = 2$. This normalisation may be helpful when combining the comparison metric with the rest of the fitness function.

Where there are several contributions to the fitness function, multi-objective optimisation seeks to find solutions that lie on the Pareto front defined by the trade-off between these two values. This is similar to the investigations in Section 3.5.2 where α, the relative importance of energy consumption and traverse time, was varied. However, multi-objective optimisation essentially seeks to optimise over a range α values simultaneously. Many GA selection operations have been proposed for use in multi-objective optimisation. They keep track of Pareto optimal solutions - those where one fitness function contribution cannot be improved without degrading the performance of another. This process would yield a number of solutions suited to different situations. The decision could then be made, possibly by an expert operator, which single solution would be best to implement in the real system.

References

[1] International Energy Agency. Key World Energy Statistics. http://www.iea.org/publications/freepublications/publication/key-world-energy-statistics-2013.html. Accessed: December 2013.

[2] Department of Energy and Climate Change, UK. Statistical press release: Digest of UK energy statistics 2013. https://www.gov.uk/government/collections/digest-of-uk-energy-statistics-dukes#2013, . Accessed: December 2013.

[3] Department of Energy and Climate Change, UK. ECUK (Energy consumption in the UK) 2013 - Transport data tables. https://www.gov.uk/government/statistics/energy-consumption-in-the-uk, . Accessed: December 2013.

[4] UK Government. Delivering a sustainable railway: white paper CM 7176. http://webarchive.nationalarchives.gov.uk/20091010004003/http://www.dft.gov.uk/about/strategy/whitepapers/whitepapercm7176/multideliversustainrailway, 2007. Accessed: October 2016.

[5] Department for Transport. Rail Technical Strategy. http://webarchive.nationalarchives.gov.uk/+/http:/www.dft.gov.uk/about/strategy/whitepapers/whitepapercm7176/railwhitepapertechnicalstrategy/pdfrailtechstrategyrts1, 2007. Accessed: October 2016.

[6] The Commission for Integrated Transport (CfIT). A comparative study of the environmental effects of rail and short-haul air travel. http://webarchive.nationalarchives.gov.uk/20110304132839/http://cfit.independent.gov.uk/pubs/2001/racomp/racomp/03.htm#31, 2001. Accessed: April 2017.

[7] http://www.nationalrail.co.uk/. Accessed: April 2017.

[8] Ning Zhao. Railway traffic flow optimisation with differing control systems. PhD Thesis (The University of Birmingham), 2013. URL http://etheses.bham.ac.uk/4725/1/Zhao13PhD.pdf.

[9] M. McClanachan and C. Cole. Current train control optimization methods with a view for application in heavy haul railways. *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, 226(1):36–47, 2012.

[10] Yihui Wang, Bing Ning, Fang Cao, Bart De Schutter, and Ton J. J. van ven Boom. A survey on optimal trajectory planning for train operations. *Proceedings of 2011 IEEE International Conference on Service Operations, Logistics and Informatics*, 19:589–594, 2011.

[11] Kunihiko Ichikawa. Application of Optimization Theory for Bounded State Variable Problems to the Operation of Train. *Bulletin of JSME*, 11 (47):857–865, 1968.

[12] I. A. Asnis, A. V. Dmitruk, and N. P. Osmolovskii. Solution of the problem of the energetically optimal control of the motion of a train by the maximum principle. *USSR Computational Mathematics and Mathematical Physics*, 25(6):37–44, 1985.

[13] P. Howlett. An optimal strategy for the control of a train. *The Journal of the Australian Mathematical Society.*, 31(4):454–471, 1990.

[14] E. Khmelnitsky. On an optimal control problem of train operation. *IEEE Transactions on Automatic Control*, 45(7):1257–1266, jul 2000.

[15] R. Liu and Iakov M. Golovitcher. Energy-efficient operation of rail vehicles. *Transportation Research Part A: Policy and Practice*, 37(10):917–932, dec 2003.

[16] Qi Wen. Energy-efficient Driving Strategies for Rail Vehicles. PhD Thesis (Imperial CollegeLondon), 2010.

[17] Phil Howlett. The optimal control of a train. *Annals of Operations Research*, pages 65–87, 2000. URL `http://www.springerlink.com/index/mg61744k2073570q.pdf`.

[18] Xuan Vu. Analysis of necessary conditions for the optimal control of a train. PhD Thesis (University of South Australia), 2006.

[19] P. G. Howlett and P. J. Pudney. *Energy-Efficient Train Control.* Advances in Industrial Control. Springer-Verlag, London, UK, 1995.

[20] B. R. Benjamin, A. M. Long, I. P. Milroy, R. L. Payne, and P. J. Pudney. Control of railway vehicles for energy conservation and improved timekeeping. *Proceedings of the Conference on Railway Engineering, Perth, Institution of Engineers Australia*, pages 41–47, 1987.

[21] Wei Song Lin and Jih Wen Sheu. Optimization of train regulation and energy usage of metro lines using an adaptive-optimal-control algorithm. *IEEE Transactions on Automation Science and Engineering*, 8(4):855–864, 2011.

[22] B.-R. Ke, C.-L. Lin, and C.-C. Yang. Optimisation of train energy-efficient operation for mass rapid transit systems. *IET Intelligent Transport Systems*, 6(1):58–66, 2012.

[23] Shaofeng Lu, Stuart Hillmansen, Tin Kin Ho, and Clive Roberts. Single-train trajectory optimization. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):743–750, 2013.

[24] H. Ko, T. Koseki, and M. Miyatake. Application of dynamic programming to optimization of running profile of a train. *Computers in Railways IX*, pages 301–312.

[25] C. S. Chang and S. S. Sim. Optimising train movements through coast control using genetic algorithms. *IEE Proceedings - Electric Power Applications*, 144(1):65–73, 1997.

[26] S. H. Han, Y. S. Byen, J. H. Baek, Tae Ki An, Su-Gil Lee, and Hyun Jun Park. An optimal automatic train operation (ATO) control using genetic algorithms (GA). *Proceedings of the IEEE Region 10 Conference, TENCON '99*, 1:360–362, 1999.

[27] J. X. Cheng, J. S. Cheng, J. Song, and P. Zhao. Algorithms on optimal driving strategies for train control problem. *Proceedings of the 3rd World Congress on Intelligent Control and Automation, 2000.*, 5:3523–3527, 2000.

[28] T. McLeod Colin Cole. Optimising train operation using simulation, fuzzy logic cruise control and evolutionary algorithms. *Fifth Asia-Pacific Industrial Engineering and Mangement Systems Conference 2004. Brisbane.*, pages 1–16, 2004.

[29] K. K. Wong and T. K. Ho. Dynamic coast control of train movement with genetic algorithm. *International Journal of Systems Science*, 35(13-14): 835–846, 2004.

[30] Y. V. Bocharnikov, A. M. Tobias, C. Roberts, S. Hillmansen, and C. J. Goodman. Optimal driving strategy for traction energy saving on DC suburban railways. *IET Electric Power Applications*, 1(5):675–682, 2007.

[31] Carmine Landi, Mario Luiso, and Nicola Pasquino. A remotely controlled onboard measurement system for optimization of energy consumption of electrical trains. *IEEE Transactions on Instrumentation and Measurement*, 57(10):2250–2256, 2008.

[32] Liu Wei, Li Qunzhan, and Tang Bing. Energy saving train control for urban railway train with multi-population genetic algorithm. *IEEE Proceedings - International Forum on Information Technology and Applications, IFITA 2009*, 2(1):58–62, 2009.

[33] S. Acikbas and M. T. Soylemez. Coasting point optimisation for mass rail transit lines using artificial neural networks and genetic algorithms. *IET Electric Power Applications*, (December 2007):172–183, 2008.

[34] Yong Ding, Yun Bai, Fang Ming Liu, and Bao Hua Mao. Simulation algorithm for energy-efficient train control under moving block system. *2009 WRI World Congress on Computer Science and Information Engineering, CSIE 2009*, 5:498–502, 2009.

[35] Qing Gu, Xiao-Yun Lu, and Tao Tang. Energy saving for automatic train control in moving block signaling system. *Proceedings: 14th International IEEE Conference on Intelligent Transportation Systems*, pages 1305–1310, 2011.

[36] Qiheng Lu and Xiaoyun Feng. Optimal control strategy for energy saving in trains under the four-aspect fixed autoblock system. *Journal of Modern Transportation*, 19(2):82–87, 2011.

[37] Yihui Wang, Bart De Schutter, Ton Van Den Boom, and Bin Ning. Optimal trajectory planning for trains under a moving block signaling system. *2013 European Control Conference, ECC 2013*, pages 4556–4561, 2013.

[38] Yihui Wang, Bart De Schutter, J. van den Boom, Ton J, and Bin Ning. Optimal trajectory planning for trains under fixed and moving signaling systems using mixed integer linear programming. *Control Engineering Practice*, 22(1):44–56, 2014.

[39] N. Zhao, C. Roberts, and S. Hillmansen. The application of an enhanced Brute Force algorithm to minimise energy costs and train delays for differing railway train control systems. *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, 228 (2):158–168, 2014.

[40] Ning Zhao, Clive Roberts, Stuart Hillmansen, and Gemma Nicholson. A Multiple Train Trajectory Optimization to Minimize Energy Consumption and Delay. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2363–2372, 2015.

[41] T. Albrecht. Reducing power peaks and energy consumption in rail tran-

sit systems by simultaneous train running time control. *Computers in railways IX*, pages 885–894, 2004.

[42] Masafumi Miyatake and Hideyoshi Ko. Numerical analyses of minimum energy operation of multiple trains under DC power feeding circuit. *EPE, 2007 European Conference on Power Electronics and Applications*, pages 1–10, 2007.

[43] Masafumi Miyatake and Hideyoshi Ko. Optimization of train speed profile for minimum energy consumption. *IEEJ Transactions on Electrical and Electronic Engineering*, 5(3):263–269, 2010. ISSN 19314973.

[44] Lixing Yang, Keping Li, Ziyou Gao, and Xiang Li. Optimizing Trains Movement on a Railway Network. *Omega*, 40(5):619–633, 2012.

[45] Hans-Georg Beyer and Bernhard Sendhoff. Robust Optimization - A Comprehensive Survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218, 2007.

[46] Wei Chen, J. K. Allen, Kwok-Leung Tsui, and F. Mistree. A Procedure for Robust Design: Minimizing Variations Caused by Noise Factors and Control Factors. *Journal of Mechanical Design*, 118(1):478–485, 1996.

[47] Yaochu Jin and Jürgen Branke. Evolutionary Optimization in Uncertain Environments - A Survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[48] Dirk V. Arnold and Hans-Georg Beyer. A Comparison of Evolution Strategies with Other Direct Search Methods in the Presence of Noise. *Computational Optimization and Applications*, 24(1):135–159, 2003.

[49] Shigeyoshi Tsutsui and Ashish Ghosh. Genetic Algorithms with a Robust Solution Searching Scheme. *IEEE Transactions on Evolutionary Computation*, 1(3):201–208, 1997.

[50] Xiang Li, Lei Li, Ziyou Gao, Tao Tang, and Shuai Su. Train Energy-efficient Operation with Stochastic Resistance Coefficient. *International Journal of Innovative Computing, Information and Control*, 9(8):3471–3483, 2013.

[51] Xiang Li, Ziyou Gao, and Wenzhe Sun. Existence of an Optimal Strategy for Stochastic Train Energy-efficient Operation Problem. *Soft Computing*, 17(4):651–657, 2013.

[52] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1 (1):67–82, 1997.

[53] Jonathan C. J. Goodwin, David I. Fletcher, and Robert F. Harrison. Multi-train Trajectory Optimisation to Maximise Rail Network Energy Efficiency Under Travel-time Constraints. *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, 230 (4):1318–1335, 2016.

[54] Lixing Yang. Personal communication (22-June), 2013.

[55] Pengling Wang, Xuan Lin, and Yuezong Li. Optimization Analysis on the Energy Saving Control for Trains with Adaptive Genetic Algorithm. *IEEE Proceedings - International Conference on Systems and Informatics (ICSAI)*, (1):439–443, 2012.

[56] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `http://www.R-project.org/`.

[57] C. Sicre, P. Cucala, A. Fernández, J. A. Jiménez, I. Ribera, and A. Serrano. A method to optimise train energy consumption combining manual energy efficient driving and scheduling. *WIT Transactions on the Built Environment*, 114(1):549–560, 2010.

[58] Lixing Yang, Shukai Li, Yuan Gao, and Ziyou Gao. A Coordinated Routing Model with Optimized Velocity for Train Scheduling on a Single-Track Railway Line. *International Journal of Intelligent Systems*, 30(1):3–22, 2015.

[59] I. Martínez, B. Vitoriano, A. Fernández, and A. P. Cucala. Statistical Dwell Time Model for Metro Lines. *Urban Transport XIII: Urban Transport and the Environment in the 21st Century*, I:223–232, 2007.

[60] William H. K. Lam, C. Y. Cheung, and Y. F. Poon. A Study of Train Dwelling Time at the Hong Kong Mass Transit Railway System. *Journal of Advanced Transportation*, 32(3):285–295, 1998.

[61] Jiaxin Yuan. Stochastic Modelling of Train Delays and Delay Propagation in Stations. PhD Thesis (Delft University of Technology), 2006. URL `http://repository.tudelft.nl/assets/uuid:caa72522-26b1-4088-afc0-59c6e5c346f6/trail_yuan_20061018.pdf`.

[62] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[63] RSSB. Engineering Driver advisory information for energy management and regulation. *T724 Stage 1 Report*, 2009.

[64] Hans-Georg Beyer. Evolutionary Algorithms in Noisy Environments: Theoretical Issues and Guidelines for Practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):239–267, 2000.

[65] RSSB. Eco-driving: understanding the approaches, benefits and risks. *T839 Report*, 2011.

[66] 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*, 224 (6):547–557, 2010.

[67] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), 2005. URL `http://www.drdobbs.com/web-development/a-fundamental-turn-toward-concurrency-in/184405990`.

[68] CPU Frequency Record. `http://hwbot.org/benchmark/cpu_frequency/`. Accessed: May 2016.

[69] Intel Product Specifications. `http://ark.intel.com/`. Accessed: May 2016.

[70] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, Fourth Edition*. Elsevier, 2007.

[71] John Nickolls and William J. Dally. The GPU Computing Era. *IEEE Micro*, 30(2):56–69, 2010.

[72] NVIDIA. Cuda C Programming Guide. `http://ark.intel.com/`. Accessed: May 2016.

[73] Henry Wong, Misel Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos. Demystifying GPU microarchitecture through microbenchmarking. *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 235–246, 2010.

[74] Erik Ruf Ray Bittner. Direct gpu/fpga communication via pci express. September 2012. URL `https://www.microsoft.com/en-us/research/publication/direct-gpufpga-communication-via-pci-express/`.

[75] Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen-mei W. Hwu. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming - PPoPP '08*, pages 73–82, 2008.

[76] R. Smith and T. S. Ganesh. The NVIDIA GeForce GTX 750 Ti and GTX 750 Review: Maxwell Makes Its Move. `http://www.anandtech.com/show/7764/the-nvidia-geforce-gtx-750-ti-and-gtx-750-review-maxwell/4`, 2014. Accessed: October 2016.

[77] ERTMS Factsheet 3: ERTMS levels. `http://www.ertms.net/wp-content/uploads/2014/09/ERTMS_Factsheet_3_ERTMS_levels.pdf`, 2014. Accessed: August 2016.

[78] Swedes unveil first ETCS Level 3 application. `http://www.railwaygazette.com/news/business/single-view/view/`

swedes-unveil-first-etcs-level-3-application.html, 2012. Accessed: October 2016.

[79] Microprocessor Standards Committee. IEEE standard for floating-point arithmetic. http://ieeexplore.ieee.org/servlet/opac?punumber=4610933, 2008. Accessed: July 2016.

[80] Rob Farber. *CUDA Application Design and Development*. Elsevier, 2011.

[81] Qizhi Yu, Chongcheng Chen, and Zhigeng Pan. Parallel Genetic Algorithms on Programmable Graphics Hardware. *Advances in Natural Computation - First International Conference*, 3612:1051–1059, 2005.

[82] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. *ACM SIGARCH Computer Architecture News*, 38(3):451–460, 2010.

[83] railML. https://www.railml.org/en/. Accessed: September 2016.

[84] Birmingham Railway Virtual Environment. http://bravesim.org/. Accessed: September 2016.

[85] Network Rail. Operations Expenditure Summary . http://www.networkrail.co.uk/browse%20documents/strategicbusinessplan/cp5/supporting%20documents/our%20activity%20and%20expenditure%20plans/operations%20expenditure%20summary.pdf?cd=5, . Accessed: September 2016.

[86] Network Rail. Rail operating centre officially opened in Manchester. `http://www.networkrailmediacentre.co.uk/news/rail-operating-centre-officially-opened-in-manchester`, . Accessed: September 2016.

[87] D. A. Stone, D. I. Fletcher, S. C. L. Koh, M. P. Foster, R. F. Harrison, A. Cruden, D. Gladwin, A. S. J. Smith, and J. Goodwin. TransEnergy - Road to Rail Energy Exchange (R2REE). `http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/N022289/1`. Accessed: October 2016.

[88] T. T. Nguyena, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6(1):1–24, 2012.

[89] J H Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[90] G Rawlins. *Foundations of Genetic Algorithms.* Morgan Kaufmann Publishers, 1991.

[91] J Koza. A Hierarchical Approach to Learning the Boolean Multiplexer Function. In G Rawlins, editor, *Foundations of Genetic Algorithms*, pages 171–192. Morgan Kaufmann Publishers, 1991.

[92] C Janikow and Z Michalewicz. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. 1991.

[93] A H Wright. Genetic Algorithms for Real Parameter Optimisation. In G Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann Publishers, 1991.

[94] D Goldberg. *Genetic Algorithms in Search, Optimisation, and Machine Learning*. Addison-Wesley, 1989.

[95] M Srinivas. Genetic algorithms: a survey. *Computer*, 27(6):17–26, 1994.

[96] F Herrera and M Lozan. Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions. *Soft Computing*, 7(8):545–562, 2003.

[97] J J Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1): 122–128, 1986.

[98] R Caponetto, L Fortuna, S Graziani, and M Xibilia. Genetic algorithms and applications in system engineering: a survey. *Transactions of the Institute of Measurement and Control*, 15(3):143–156, 1993.

Appendix I

Introduction to Genetic Algorithms

General principle

Genetic algorithms (GAs) are a type of heuristic optimisation strategy, pioneered by Holland [89], which mimic evolution by natural selection. In natural selection, variation in a population results in some individuals being more fit for their environment than others. These individuals are therefore more likely survive, reproduce, and pass their chromosomes onto the next generation; the expected outcome being that, successive generations will inherit characteristics which make them more suited to the environment than their ancestors. The analogy can be made that: *evolution* (the algorithm) seeks to finds the *fittest* (optimal) *chromosome* (solution) for the *environment* (objective function). Figure I.1 shows the main stages in a genetic algorithm.

An important decision when applying a GA to a problem is how to suitably codify the candidate solutions (here after referred to as chromosomes). Traditionally, chromosomes have been represented as fixed length binary strings.

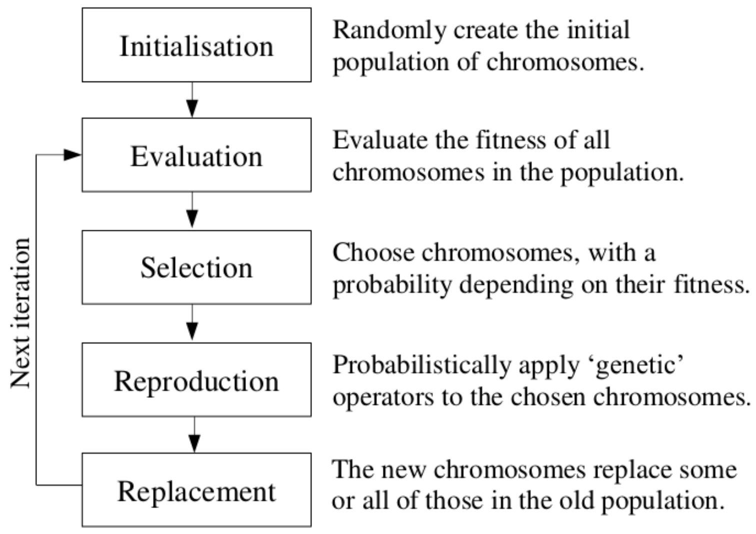

Initialisation
Randomly create the initial population of chromosomes.
Evaluation
Evaluate the fitness of all chromosomes in the population.
Selection
Choose chromosomes, with a probability depending on their fitness.
Reproduction
Probabilistically apply 'genetic' operators to the chosen chromosomes.
Replacement
The new chromosomes replace some or all of those in the old population.
Next iteration

Figure I.1: General description of a genetic algorithm[90].

However, this may limit the accessible solution space when the solution size or shape is not known in advance[91] so many other representations, such as floating points [92] and vectors of real numbers [93] have also be considered for particular systems.

Worked example

The following worked example is instructive for illustrating the different operations used during the GA, and also is used to introduce the important concepts of *schema* and the *building block hypothesis*.[89, 94]

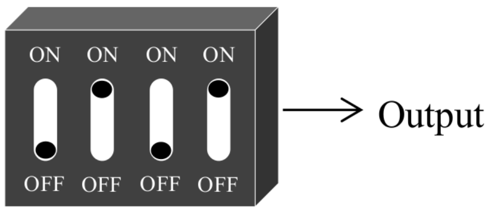


Figure I.2: Example system to be optimised.

- **Problem:** Maximise output from a black box with four toggle switches (figure I.2).
- **Representation:** Each chromosome can be represented as a binary string of length four, where 1 = switch on, and 0 = switch off. The switch combination shown in figure I.2 would be encoded as 0101.
- **Initiation:** Create a (pseudo)random population of chromosomes.

  Chromosomes = 1010, 0011, 1100, 0101
- **Evaluation:** Evaluate each chromosome (using the black box).

  Output = 10, 3, 12, 5 (respectively)

Schemata describe subsets of strings which have similarities at certain positions. Using * as a wild card (0 or 1), the schema 1**0 represents all 4 (2^2) different 4 bit binary strings with a 1 in the first position and a 0 in the fourth position. The order of a schema is the number of fixed position it contains; we can see, that in the example, that the two highest scoring strings are both instances of the 2 nd order schema 1**0. However, it would be equally true to say that 1100 is an instance of 1*** or ***0 or 11*0 etc. In-fact, any n digit binary string will be an instance of 2^n different schemata simultaneously. For each set of set of k fixed positions there are 2^k different competing schemata, so a GA can be seen as "simultaneously, though not independently, attempting to solve all the 2^n schema competitions and locating the best schema for each set of fixed positions."[95]

- **Selection:** Probabilistically choose the chromosomes to undergo reproduction operations and enter the next generation. A simple selection procedure is roulette-wheel selection, where the probability (P) of choos-

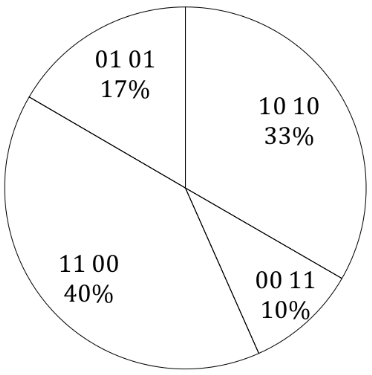


Figure I.3: An example of a roulette-wheel which must be 'spun' during selection. Fitter chromosomes are more likely to be selected to undergo reproduction.

ing a chromosome (i) is related to its fitness (F). e.g. $P(i) = F_i/F_{total}$. This idea is illustrated in figure I.3.

- **Reproduction:** There are three main types of reproductive operation: reproduction, crossover, and mutation (see figure I.4).
  - Reproduction is the direct duplication of a chromosome - it conserves the exact genetic material of the parent chromosome.
  - Crossover takes two chromosomes and swaps parts of them creating two new chromosomes - it reorganises genetic material from two parent chromosomes, *exploiting* the knowledge already obtained.
  - Mutation makes random changes to a chromosome (bit-inversion in this representation) - it creates new genetic material, which may have been lost from the population, *exploring* new areas of the search space.

There is a balance, between *exploring* and *exploiting* the knowledge encoded in the genetic material, which must be maintained to avoid premature convergence[96]. Optimisation of the control parameters must be performed, and may be turned during the GA process[97].

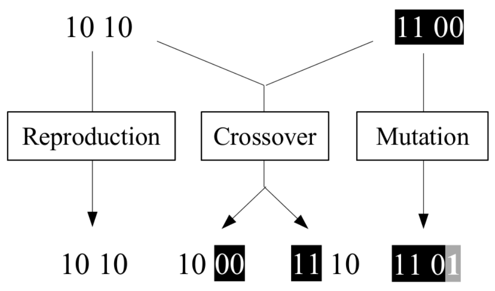


Figure I.4: Illustration of GA reproduction operations.

Selection increases the likelihood that high fitness schemata will become more prevalent in each consecutive generation. However, the longer the schemata the higher the chance it will be disrupted by crossover or mutation, so it is high fitness schemata with short defining lengths that actually grow in prevalence. These known as building blocks, and the *building block hypothesis* assumed that the juxtaposition of good building blocks leads to a good overall string.[95]

- **Replacement:** There are two main procedures for introducing chromosomes into the next generation. A generational procedure will create an entirely new population using the chromosomes produced by applying genetic operators, whereas a steady-state method keeps the same population, but replaces a few chromosomes at a time.

Finally, it has been noted that, since GAs optimise for high-performance schemata within the whole population, individual chromosomes are of little importance so some sort of local search should be applied to the entire population to find the best solution.[98]

Appendix II

Input data for base case in Chapter 3

This appendix contains the input data from Yang et al. [44], first used in Chapter 3 and then built on in subsiquent chapters. The network topology is shown in Figure II.1. All three lines are single track (meaning trains can not pass each other) and 30 km long.

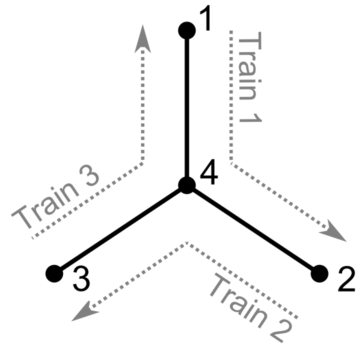


Figure II.1: Illustration of network N1 showing its topology and the routes of the train journeys. Nodes represent stations and edges single track lines.

The movement of three trains was modelled on the network, with each train making two journeys separated by a stop for operations (e.g. boarding and alighting of passengers) at the centeral station 4. These routes are shown in Table II.1 along with other train specific parameters.

Table II.1: Information unique to each train

Train Index	Route	Operation time (s)	Weight (t)	Target E per journey (kWh)	Target T per jouney (s)
1	$1 \to 4 \to 2$	30	665	800	650
2	$2 \to 4 \to 3$	20	600	800	650
3	$3 \to 4 \to 1$	25	565	800	620

The maximum speed limit for all the lines was 300 km/h. However, on each of the three lines a section of reduced speed limit was defined. Route $1 \to 4$ traverses the same track as route $4 \to 1$ but in the opposite direction. This means that the same section of reduced speed limit is encounted at a different position when traversing the line in differnt directions. For clarity Table II.2 gives the position of reduced speed limit as they are encounterd on each line in each direction.

Table II.2: Speed limits on each line

Line	Speed limit (km/h)	Start of limit (km)	End of limit (km)
(1, 4)	200	15	20
(4, 1)	200	10	15
(2, 4)	150	10	13
(4, 2)	150	17	20
(3, 4)	230	20	23
(4, 3)	230	7	10

The traction, resistance and braking force (in kN) of all three trains were identicle. The traction force was given by:

$$traction force = \begin{cases} 360 & if 0 \leq v \leq 180 km/h \\ 360 - \frac{6}{7}(v - 180) & if 180 \leq v \leq 300 km/h \end{cases} \tag{II.1}$$

where v is the velocity of the train in km/h. Although Equation II.1 was stated in [44] section 3.1.2 of this thesis discussed the fact that it appears that the actual traction force implemented was given by:

$$traction force = 360, if 0 \leq v \leq 300 km/h \tag{II.2}$$

The resistance force was given by:

$$resistance force = 11.4 + 0.101v + 0.001269v^2 \tag{II.3}$$

The braking force was given by:

$$braking force = \begin{cases} 300 - 0.2v & if 0 \leq v \leq 100 km/h \\ 280 - 1.2v(v - 100) & if 100 \leq v \leq 200 km/h \\ 160 - 0.5v(v - 200) & if 200 \leq v \leq 300 km/h \end{cases} \tag{II.4}$$

Appendix III

Roulette wheel selection on a GPU

The optimisations described in chapters 3 and 4 biased-random selection of individuals has used the roulette wheel selection scheme described by Yang et al. [44]. After ranking the population by their fitness scores (from best to worst), each individual, $\underline{\boldsymbol{X}}_i$ was assigned a 'rank-based evaluation value' given by:

$$Eval(\underline{\boldsymbol{X}}_i) = \alpha(1-\alpha)^{(i-1)} \tag{III.1}$$

where $\alpha \in (0,1)$ is a pre-determined parameter (implemented as 0.05). These values were then accumulated over a vector $\underline{W}$ such that $W_0 = 0, W_i = \sum_{j=1}^{i} Eval(\underline{\boldsymbol{X}}_i)$, where $i = 1, 2..., \texttt{pop\_size}$. For each individual selected, a random number $t \in [0, W_{\texttt{pop\_size}})$ was generated and a linear search performed on vector $\underline{W}$ until the rank i was found, i.e., $t \in [W_{i-1}, W_i)$.

There are several reasons why Yang's implementation of roulette wheel selection is expected to be inefficient on a GPU. These reasons mostly result from its use of a look-up table, which will lead to increased memory transactions (scaling proportional to `pop_size`) as well as thread divergence. One obvious solution to this would be to use the inverse cumulative probability function for rank-based selection. For the rank-based evaluation value used (Equation (III.1)), this turns out have a relatively simple form.

The cumulative probability of solution with rank i (in the sorted population) is given by:

$$cumProb(i) = \frac{\sum_{j=1}^{i} \alpha(1-\alpha)^{(j-1)}}{\sum_{j=1}^{\texttt{pop\_size}} \alpha(1-\alpha)^{(j-1)}} \tag{III.2}$$

which simplifies to

$$cumProb(i) = \frac{\sum_{j=0}^{i-1} (1-\alpha)^{j}}{\sum_{j=0}^{\texttt{pop\_size}-1} (1-\alpha)^{j}} \tag{III.3}$$

In general, for $x \neq 1$ it is true that

$$\sum_{j=0}^{n} x^j = 1 + x + ... + x^n = \frac{x^{n+1} - 1}{x - 1} \tag{III.4}$$

So, Equation (III.3) can be re-writen as:

$$cumProb(i) = \frac{(1-\alpha)^i - 1}{(1-\alpha)^{\texttt{pop\_size}} - 1} \tag{III.5}$$

To find the inverse of the probability distribution function we must solve for i which gives:

$$i = \frac{log(cumProb(i) \cdot ((1-\alpha)^{\texttt{pop\_size}} - 1) + 1)}{log((1-\alpha))} \tag{III.6}$$

so the rank, i, of a selected individual can be found using the equation:

$$i = ceil\left(\frac{log(t \cdot c_1 + 1)}{c_2}\right) \tag{III.7}$$

where, $c_1 = (1-\alpha)^{\texttt{pop\_size}} - 1$, $c_2 = log(1-\alpha)$, $t \in [0, 1)$ is a (pseudo)random number, and *ceil* is an operator which rounds up to the nearest integer.

Using the relation in Equation (III.7) to select individuals should be faster than the look-up table method as only two constants, c_1 and c_2, need to be loaded from memory (rather than an array of `pop_size` $+ 1$ elements). Also, the process of looking up the values from this array is replaced by a single calculation which avoids many memory accesses as well as thread divergence within each warp.

Appendix IV

Basic kinematics of trains

IV.0.5 Physics of train motion

In order the investigate optimisation techniques it is necessary to model the movement of trains. At the most fundamental level, the motion of a train depends on Newton's second law:

$$F = ma \tag{IV.1}$$

where F is the resultant force acting on the train, m is the mass of the train, and a is the acceleration of the train. In this thesis the components of the result force are approximated as:

$$F = F_{control} - F_{resistance} - F_{gradient} \tag{IV.2}$$

Consistent with most train train trajectory optimisation literature, the resistance force on the train is given by the 'Davis Equation':

$$F_{resistance} = a + b|v| + cv^2 \tag{IV.3}$$

where v is the velocity of the train and a, b and c are empirically derived coefficients which vary between different rail vehicles. The component of the force due to the gradient of the track is given by:

$$F_{gradient} = mgsin\theta \tag{IV.4}$$

where g is the gravity of Earth and θ is the gradient of the slope above horizontal.

The component of force due to train control can be used to calculate the traction energy consumption of the train. The work done by a force is given by:

$$E = Fx \tag{IV.5}$$

However, neither the traction nor the regeneration systems of trains are 100% efficient so the total energy consumed is given by:

$$E = \frac{F_{traction}}{\gamma_{traction}} + \gamma_{regen} F_{braking} \tag{IV.6}$$

where $\gamma_{traction}$ and γ_{regen} are the efficiencies of traction and regenerative braking respectively and are in the ranges [0,1).

IV.0.6 Solving the equations of motion

The movement of trains can be modeled using the differential equations:

$$a = \frac{\delta v}{\delta t} \tag{IV.7}$$

$$v = \frac{\delta x}{\delta t} \tag{IV.8}$$

where a is acceleration, v is velocity and x is position.

Assuming constant acceleration over each time step (Δt) Equation (IV.7) can be solved by integration to give Equation (IV.11). This avoids the need to solve the equations of motion directly, which would be infeasible given the number of distance based variables (e.g. gradient, driving style).

$$\int_t^{t+\Delta t} a dt = \int_t^{t+\Delta t} dv \tag{IV.9}$$

$$a\Delta t = v_{t+\Delta t} - v_t \tag{IV.10}$$

$$v_{t+\Delta t} = v_t + a\Delta t \tag{IV.11}$$

Similarly, substituting Equation (IV.8) into Equation (IV.11) and solving by integration gives:

$$x_{t+\Delta t} = x_t + v_t\Delta t + \frac{1}{2}a\Delta t^2 \tag{IV.12}$$

Equations (IV.11) and (IV.12) allow the velocity profiles of trains to be found by accumulating calculations over many small time steps. This is the method underlying simulation in models G1 to G5. Traction energy consumption of the trains in these models was calculated in a similar way by combining Equation (IV.20) and (IV.12) to give:

$$E_{t+\Delta t} = E_t + F(v_t\Delta t + \frac{1}{2}a\Delta t^2) \tag{IV.13}$$

Alternatively, train motion can be given in terms of small distance steps (Δx). Equation (IV.7) and (IV.8) can be combined to give the differential equation:

$$\frac{v}{a} = \frac{\delta x}{\delta v} \tag{IV.14}$$

Assuming constant acceleration over Δx Equation (IV.14) can be solved by integration to give Equation (IV.17).

$$\int_{x}^{x+\Delta x} \frac{v}{a} dv = \int_{x}^{x+\Delta x} dx \tag{IV.15}$$

$$\frac{v_{x+\Delta x}^2 - v_x^2}{2a} = \Delta x \tag{IV.16}$$

$$v_{x+\Delta x} = \sqrt{v_x^2 + 2a\Delta x} \tag{IV.17}$$

Similarly, over the small distance step (Δx) Equation (IV.8) can be solved by integration to give:

$$a(t_{x+\Delta x} - t_x) = v_{x+\Delta x} - v_x \tag{IV.18}$$

Substituting a from Equation (IV.16) and solving for $t_{x+\Delta x}$ gives:

$$t_{x+\Delta x} = t_x + \frac{2\Delta x}{v_x + v_{x+\Delta x}} \tag{IV.19}$$

Equation (IV.17) and (IV.19) are used to simulate the velocity profiles of trains in model G6. Traction energy consumption of the trains in this models were calculated using:

$$E_{x+\Delta x} = E_x + F\Delta x \tag{IV.20}$$