Dynamic Scheduling of Trains in Densely Populated Congested Areas

Final Report

METRANS Project

February 4, 2011

Principal Investigator:

Maged M. Dessouky, Ph.D.

Graduate Student:

Shi Mu

Daniel J. Epstein Department of Industrial and Systems Engineering

University of Southern California

Los Angeles, California

Disclaimer

Abstract

In the U.S., freight railways are one of the major means to transport goods from ports to inland destinations. According to the Association of American Railroad's study, rail companies move more than 40% of the nation's total freight. Given the fact that the freight railway industry is already running without much excess capacity, better planning and scheduling tools are needed to effectively manage the scarce resources, in order to cope with the rapidly increasing demand for railway transportation. Freight train scheduling and dispatching is one important aspect of freight railroad management. This research tries to develop algorithms for static and dynamic scheduling of freight trains. Two optimization based algorithms are first proposed to solve the static train scheduling problem for small rail networks. The proposed LtdFlePath and GA+FixedPath algorithms are able to outperform two existing heuristics, PureGA and Greedy algorithm, in terms of railway total train delay. And the CPU solution times of the proposed heuristics are within a reasonable time constraint. Then two decomposition based heuristics, the Decomp and Parallel algorithm, are developed to solve the train scheduling problems for larger networks. Both algorithms significantly outperform existing algorithms. Finally, we move to dynamic scheduling of trains. We present one heuristic which solves the dynamic scheduling problem for both small and large networks. This Dynamic algorithm is able to reduce delay by at least 40% of existing algorithms on representative rail scenarios.

Disclosure

The project was funded in entirety under this contract to California Department of Transportation.

Acknowledgement

We would like to thank METRANS for funding this research.

Contents

List of Tables

List of Figures

1 Introduction

The demand for various methods of transportation has increased due to the emerging global economy. Imported goods from other countries usually enter the United States through ports and then transported inland. Every year there are more than 100 million tons of goods transferred through the Ports of Los Angeles and Long Beach and this number will double by 2020 (Leachman, 2002). Train transportation is a cost effective way to move cargo from the ports to distant inland destinations. According to the Association of American Railroad's study, rail companies move more than 40 percent of the nation's total freight. As the total quantity of freight increases, by year 2020, the railroad industry expects to see demand increases as much as double the amount the industry is experiencing today.

Given the fact that the US freight railroad industry is already running without much excess capacity, the freight railroad industry has to either expand its infrastructure or manage its current operations more efficiently to meet the anticipated increase in demand. It is extremely expensive to build more rail tracks and in some places like Los Angeles County, due to the limited space, it is almost impossible to expand the current track. Better planning and scheduling methodologies become an effective solution to the problems caused by increasing transportation demand under tight capacity constraint.

Freight railroad management is a complicated problem as a whole. Thus, the overall management problem is usually decomposed into several subproblems. They are:

- Crew scheduling. The crew assignments to each district, line and train need to be optimized such that the crew costs and the delays due to the unavailability of crews are minimal.
- Blocking problem. The shipments are classified into different blocks according to their origins and destinations. The blocking plans for the shipments are optimized

to reduce the total transportation and handling cost.

- Yard location. This problem is closely related to the blocking problem. The yards are the places where the old cargo blocks are re-distributed into new blocks and transferred to different trains.
- Train routing. Given the assignment of the cargo blocks, the routes and departure times of the trains are determined to minimize transportation cost and delays.
- Locomotive scheduling. Once the train routes are determined the locomotives are assigned to the trains. The locomotive assignments have to satisfy the pulling-power requirement and constraints like fueling and maintenance constraints.
- Train scheduling and dispatching. Operational scheduling of the trains determines in detail the sequence of the tracks the train travels and the specific locations on the track where the trains need to stop to wait for the tracks ahead to be cleared. The objective of the scheduling and dispatching problem is to safely guide the trains through the network to their destinations so that the delays and deviations from the planned timetable of the trains are minimized.

The most studied subproblem in the literature is the blocking problem. This report focuses on the problem of freight train scheduling and dispatching. Recently, the train scheduling problem has received increased attention by academic researchers and industry alike due to two reasons. First, the emergence of affordable computers with very fast processing speed makes it possible to solve relatively large scale train scheduling problems using a computer-based optimization model. Second, due to the increased usage of rail as a mode of transportation, more and more trains are traveling on limited track resources. Thus a good schedule for the trains becomes vital in order to prevent the melt-downs of the rail network. In the case where trains are not so dense on the network, a not-so-well-designed schedule might not perform much worse than the optimal schedule. However when the networks are close to saturation, a well designed schedule can make a significant difference in minimizing the delay.

In urban areas like Los Angeles County, the trackage configurations are usually very complex, compared to rural areas where most trackage configurations are single

tracks with sidings or double tracks. A typical complex network contains triple tracks and complex junction intersections. The problem of finding the optimal deadlock-free dispatch times that minimizes the delay for trains in such a general network is known to be NP-hard (Garey & Johnson, 1979).

As opposed to passenger train scheduling, freight train scheduling might need a different approach. The passenger train schedules are relatively static and cyclic. Passenger train operators normally spend months before their operations to develop a robust schedule. And this schedule is executed cyclically on a day-to-day basis. In passenger train scheduling the quality of the schedule is the most important factor, since the time to develop the schedule is not constrained. Whereas in freight train scheduling, the scheduling procedure is initiated very close to the time of the departure of train. The short scheduling time is due to the fact that the departure time of the train is mainly initiated by the completion of loading the containers onto the train. And the ready time of the containers to be loaded is very difficult to predict. This ready time depends on the operations of the ports and the arrival time of the cargo ships. In most cases, the departure times of trains are known just one day before its departure. And it is not unusual that freight trains depart without schedules beforehand. Hence, freight train scheduling focuses on both the solving time and the solution quality.

This project focuses on developing optimization-based heuristics for both static and dynamic freight train scheduling on a complex train network. In order to measure the performance of the algorithms compared to the optimal solution, a small network was constructed and the optimal or close to optimal solutions were obtained to construct a benchmarking tool for any proposed algorithms.

The rest of the report is organized as follows. In Section 2, a literature review of the train scheduling problem is presented. Section 3 summarizes our main research accomplishments. Section 4 formally introduces the problem formulation of static scheduling. Section 5 compares various existing and proposed algorithms for solving a static scheduling problem for a relatively small to medium size but not necessarily simple network. A heuristic is proposed to solve the static scheduling problem for

relatively large scale networks in Section 6. In Section 7, we move to dynamic scheduling of trains and present a heuristic which is based on sequential optimization. The conclusion of this project and future research plan are described in Section 8.

2 Literature Review

The train scheduling problem can be categorized into static and dynamic scheduling. For static scheduling, we have information of all the trains before solving the problem. This information includes train arrival times, train lengths, train speeds and so on. For dynamic scheduling, we may only know the information of the trains when they enter the network. The schedule of the newly entered train is based on the information of the trains currently in the network. All the information of later arriving trains is unknown.

The static train scheduling problem can be divided into two sub-categories according to two different time perspectives: off-line timetabling and real time dispatching. Off-line timetabling aims to develop detailed timetables for the trains before the departure of the train. The trains travel on the rail network according to the timetables. The off-line timetabling problems can be further categorized according to the type of the trains. Passenger train scheduling belongs to the domain of tactical scheduling and freight train scheduling is often referred as operational scheduling in the literature. The trains may not always follow the off-line timetables. There are many possible incidents that can cause the extra delay of trains at some point in the network. When a single train deviates from its original timetable, other trains in the network will be influenced. The delay propagates through the network. The purpose of real time dispatching is to minimize the unplanned delays in the network and to restore the railway traffic to the planned timetables.

Cordeau et al. (1998) published a comprehensive survey paper on both train routing and off-line scheduling. More recently, Caprara et al. (2006) presented a

review on passenger railway optimization which focused more on the European environment. On the other hand, Ahuja et al. (2005) reviewed network models for railroad planning and scheduling. Their review focused on the freight railroad in North America. Next we review the relevant literature by classifying them into three categories: tactical scheduling, operational scheduling and real time dispatching.

2.1 Tactical Scheduling

Tactical scheduling is usually carried out several months before the actual operations. The objective of tactical scheduling is to design the optimal schedules that satisfy the demand of the various stakeholders for the trains (in most cases, passenger trains). Models and algorithms for tactical scheduling usually perform optimal slot allocations for each route or even block section without a strict time limit of computation (D'Ariano 2008). Scheduling is usually done on a large traffic network and once the optimal schedules are obtained, it can be used for several years.

Szpigel (1973) describes one of the earliest scheduling models. The model solves the scheduling problem on a single track with meet/pass points. Szpigel formulates the problem similar to the job shop problem with a branch and bound approach used to solve the model. A single track railroad in Brazil is used as the test case.

Jovanovic and Harker (1991) propose a system called SCAN for tactical freight train scheduling. The system is able to design robust train schedules under stochastic operational conditions. The system works with single and double tracks and develops daily schedules. An initial schedule is needed and the feasibility of this initial schedule is verified by solving a linear MIP problem, using a branch and bound procedure. The train movements and interactions are modeled using a simulation approach. The infeasible schedules are modified into feasible schedules by a series of automatic updates. The authors claim that the implementation of the SCAN system at a major U.S. railroad has helped the company rethink the role of careful scheduling in light of increasing competitive pressures.

Brannlund et al. (1998) introduce an optimization model which maximizes the profit associated with running different types of services at specific times. The problem is approached by formulating a large integer programming problem which is solved by Lagrangian relaxation where the track capacity constraints are dualized. For each single train, the problem is decomposed into a shortest path problem on a time-space network. Trains are dispatched through the network sequentially and a priority list is used to resolve the conflicts. Computational experiments show that feasible solutions can be obtained in a reasonable amount of time and feasible solutions are within a few percent of the lower bound.

Capra, Fischetti, Toth (2002) solve the train scheduling problem of a single, one-direction track connecting two major stations with several intermediate stations in between. They formulate a linear integer programming model based on a graph theoretic formulation and apply Lagrangian relaxation to find a good solution. The relaxation is embedded in a heuristic algorithm which consists of three parts: constructive, refining and fixing. The algorithm is implemented and tested using actual data and highly congested instances.

Ghoseiri et al. (2004) develop a multi-objective optimization model for the passenger train-scheduling problem on a complex railroad network. The objective includes minimization of both the fuel consumption and the total passenger-time. The solution procedure consists of two steps. The Pareto frontier is computed and then the multi-objective optimization is performed using a distance-based method.

Zhou and Zhong (2005) formulate an integer programming problem to solve the multi-objective train scheduling problem. The objectives the authors consider are: (1) minimizing expected waiting time for high speed trains and (2) minimizing total traveling time for high-speed and medium-speed trains. A branch-and-bound algorithm with dominance rules is developed to compute Pareto-optimal schedules. A beam search algorithm with utility evaluation rules is proposed to generate a representative set of non-dominated solutions. The proposed algorithm is tested on a double-track intercity passenger corridor between Beijing and Shanghai.

Zhou and Zhong (2007) optimally solve a single-track train timetabling problem.

The scheduling problem is formulated as a generalized resource-constrained project scheduling problem. The track segments and stations are modeled as resources. A branch and bound procedure is used to find the optimal solution. Three procedures are proposed to reduce the search space: a Lagrangian relaxation based lower bound rule is used to relax segment and station headway capacity constraints, an exact lower bound rule to estimate the least train delay, and an upper bound constructed by a beam search heuristic method. Numerical experiments are conducted to compare the performance of the proposed three search space reduction procedures.

2.2 Operational Scheduling

The biggest distinction between operational scheduling and tactical scheduling is the duration of the scheduling phase. As opposed to tactical scheduling, which is done without a strict time constraint, operational scheduling has a much shorter duration. A typical operational schedule is created in a few hours. It is common that the train leaves the station without a schedule. Operational scheduling is mostly used in the North American and Australian freight rail industry. Due to the time constraint of the scheduling phase, exact optimal solution procedures are not applicable. Thus heuristics are the most common approaches for operational scheduling.

Kraay et al. (1991) consider a scheduling and pacing problem which minimizes both the fuel consumption and travel delays. Instead of assuming trains run at a fixed speed, the model considers variable speeds of the trains. A nonlinear mixed integer program is formulated with a convex objective function. Branch-and-bound and a rounding heuristic are proposed to solve the scheduling and pacing problem. Numerical experiments show that the fuel consumption can be reduced in the order of 5% and the standard deviation in train arrival times can decrease by 19% using this approach.

Kraay and Harker (1995) propose a model to provide a link between tactical and operational scheduling. They propose a non-linear mixed integer programming model

to optimize the freight train schedules in real-time. The current positions and relative importance of each train are part of the input to the model. The solution of the model includes the target time of each train at every important location along the path. The solution process first determines the integer variables of the model. Then the sub-problem which only has continuous variables is solved. The efficiency and efficacy of this algorithm is justified by testing it on a North American railroad example.

Carey and Lockwood (1995) describe a mathematical model to dispatch trains on a one-way single line with sidings and stations. A heuristic is proposed to solve the problem by dispatching trains one by one. The number of integer variables is significantly reduced. Thus the sub-MIP problem can be solved using commercially available optimization software. Several procedures that are based on experienced human dispatchers are also proposed to reduce the solving time. Carey (1994a) extends the previous model by embedding a route selection mechanism in the mathematical model. Carey (1994b) then further expands the model to take two-way tracks into consideration. The same heuristic algorithm proposed in Carey and Lockwood (1995) still applies in such networks.

Huntley et al. (1995) develop a system called computer-aided routing and scheduling system (CARS) for CSX transportation. The system optimizes the routing and scheduling problem interactively. The CARS system uses simulated annealing to perform a global search on the minimum cost solution.

Higgins et al. (1996) formulate a non-linear mixed integer program to solve the scheduling problem on a long single-track line. The lower and upper limit of the train velocity on each track segment is considered. The objective is to minimize both the fuel consumption and overall tardiness. The train priorities, current train delays and expected remaining delays of the trains are criterions for conflict resolution. In a following paper, Higgins and Kozan (1997) extend their work to simultaneously decide the number and locations of the sidings and the optimal schedule for a single-track line.

Ping et al. (2001) use genetic algorithms to adjust the departure order of the

trains on a double track corridor. Simulation results of a case study on Guangzhou-Shenzhen high-speed railway are presented.

Lu et al. (2004) introduce train acceleration and deceleration rates into the scheduling model. The model also considers a very complex trackage configuration with multi-tracks and complicated crossings. A simulation model is developed and a greedy construction heuristic is used to dispatch the trains in the simulation model. The simulation model is tested on a real network from Downtown Los Angeles to the Eastern Inland Empire area. The results obtained from the simulation model are validated against actual train running times, and are found to be within a few percentage of the actual times.

Dorfman and Medanic (2004) propose a discrete-event model to schedule traffic on a railway network. This approach is fundamentally different from the mathematical programming methods. The discrete-event model is computationally efficient and generates near optimal schedules with respect to a number of time-of-travel-related criteria.

Wegele and Schneider (2004) propose an algorithm for fast construction of timetables. Branch and bound is used to obtain the initial solution and a genetic algorithm is used to iteratively improve the solution. The objective of the problem is to minimize the annoyance to passengers.

Sahin et al. (2004) propose to model the train dispatching problem as a multi-commodity flow problem on a space-time network. Time is discretized with respect to equal-length time periods. Most of the practical constraints can be considered in the model without changing the model structure. An integer programming based heuristic is proposed to solve the problem. The proposed solution procedure is tested on extensive numerical experiments. The results of the IP-based heuristic are compared to two other heuristics proposed by the authors: a simulation-based construction heuristic and a greedy enumeration heuristic. The solution quality of the IP-based heuristic is very good and the solving time is also competitive compared to the other heuristics.

Dessouky et al. (2006) propose a branch and bound procedure to solve the

dispatching problem for a complex rail network. Adjacent propagation and feasibility propagation is used to reduce the search space of the branch and bound procedure. The branch and bound procedure is guaranteed to find the optimal solution. The proposed solution procedure is tested on a portion of the rail network in Los Angeles County. The proposed solution is able to significantly reduce the number of nodes to be explored compared to the number of nodes explored by the CPLEX solver on the same problem.

2.3 Real Time Dispatching

Off-line timetables produce a robust schedule to execute. However, in real railway operations, perturbations can happen. Significant perturbations would create serious delay propagation. Technical failures might be one cause of the perturbation. Unavailability of the crew and severe weather conditions are also common causes of the unscheduled delay. When deviations from the timetable occur, the dispatcher in the traffic control center needs to quickly solve the unplanned conflicts between the trains. Common resolutions include changing the train order at busy junctions, changing the dwell time at the stations and changing the train speeds. Sometimes rerouting of the trains is also considered. The problem is usually solved regionally, due to the time constraint. The output of the real time dispatching is a locally optimal solution and in many cases, just a feasible solution.

Sahin (1999) solves the dispatching problem by proposing an algorithm that is constructed on immediate inter-train conflict resolution. The immediate conflict is solved by choosing the solution that causes less total consequential delay on the network. The algorithm considers the effects of potential conflicts by using a look-ahead method. Numerical examples show the heuristic proposed is able to produce solutions that are as good as the exact solutions. And the heuristic takes less than one percent of the time of the exact solution method.

Adenso-Diaz et al. (1999) use a heuristic to solve the MIP formulation of the

real time dispatching problem. The heuristic is based on backtracking and reduces the search space by elimination of certain branches which are determined to not generate good solutions. The quality of the solution is determined by the priority of each service, the passengers transported and the delays of the trains. A tool based on a proposed heuristic has been implemented for the Spanish National Railway Company.

Mascis et al. (2002) propose to solve the real time scheduling problem using an alternative graph formulation which is generalized from the job shop scheduling problem with blocking and no-wait. They show several key properties, from the literature on the job shop with unlimited buffers, do not hold in the blocking and no-wait cases. And some ideas used to develop the branch and bound algorithm can be easily extended from the literature.

D'Ariano et al. (2007) also formulate the dispatching problem as a huge job shop scheduling problem with no-store constraints. They use the branch and bound approach to solve the problem. They develop both the dynamic implication and static implication rules to reduce the search space. To assess the performance of the proposed procedure, they implement two local simple dispatching rules which simulate the typical behavior of a human dispatcher, and a greedy heuristic based on global information. The computational experiments, which are based on a bottleneck area of the Dutch railway network, show very promising performance of the proposed algorithm in finding a near optimal solution within short computation times.

Tornquist (2007) formulates and solves a re-scheduling problem on a geographically large and fine-grained railway network. The author formulates the problem as a MILP and proposes four strategies for solving the problem. Strategy 1 allows swapping of the track but not for orders, whereas strategy 2 allows swapping tracks and also implicit order changes. Strategy 3 allows for a certain number of order swaps. Finally strategy 4 allows all possible changes. Experiments on the network of the South traffic district in Sweden show that strategy 3 appears to perform well with respect to computation time and solution quality.

Rodriguez (2007) uses constraint programming to solve the routing and scheduling of trains travelling through a junction. Three propagation mechanisms are

used to prune many non-feasible decisions. Numerical results based on a real junction North of Paris show that the proposed method can significantly improve the decisions applied by the operator, and the computation time is less than the time required to apply the solution in real conditions.

Real time dispatching is different from operational scheduling in the sense that real time dispatching has even tighter constraints on the computation time, normally less than three minutes. The scale of the problem is smaller in real time dispatching, and in most cases, local near optimal solutions are satisfactory.

2.4 Gap

Most of the literature in the domain of operational scheduling focus on small scale rail networks. Optimal procedures have been developed for small networks with various restrictions. Typical simplification of the network includes one way travelling and single line railway configuration. This project explores optimal procedures for solving scheduling problems on small networks without simplifying the network. The algorithms developed for small networks also serve as benchmarking tools for algorithms developed for larger networks. Existing procedures in the literature for solving scheduling problems on large networks use either simulation or a heuristic approach. This project proposes a decomposition approach for large scale static scheduling problems. The large problem is decomposed into smaller sub-problems and then the sub-problems are solved by the procedures proposed for smaller networks. For dynamic train scheduling, the literature provides very limited references. This project tries to solve the dynamic scheduling problem using a sequential optimization approach.

3 Research Accomplishments

As mentioned previously, for static scheduling, most exact methods for small networks in the literature are done by assuming a simple structure of the network. For large scale networks, most of the research in the literature is based on pure simulations or simple heuristics. And there is very limited existing work on dynamic scheduling of trains in complex rail networks. During the one year of this research project, our efforts were mainly concentrated in addressing these gaps. The tasks accomplished can be broadly classified as follows.

1. Develop two optimization based static scheduling algorithms for small complex rail networks. Use these two algorithms to benchmark developed heuristics for larger networks.
2. Develop two decomposition based static scheduling algorithms for large complex rail networks. The large scheduling problem is decomposed into smaller sub-problems. Each sub-problem is a standard train scheduling problem that can be solved by algorithms proposed for small networks.
3. Propose an algorithm for dynamic scheduling on both small and large networks. This dynamic scheduling algorithm uses the idea of sequential optimization. The performance of this algorithm is benchmarked with the performance of one simulation based heuristic.

4 Problem Formulation

4.1 Network Construction

The objective of operational scheduling for freight trains is to move each train from its origin to its destination as fast as possible so that the total delay of all the

trains are minimized. The delay of a train is defined to be the difference between its actual traveling time on the track and the shortest traveling time possible assuming there are no other trains in the network. In order to formulate the problem mathematically, the actual rail network needs to be translated into nodes and arcs. The network construction method in Lu et al. (2004) is adopted. A node denotes a train track segment, a station or a junction. Different nodes could have different speed limits imposed on it. An arc denotes the linkage between nodes. Normally, the length of a junction node is zero, so is the arc element. Each track node has a capacity of one, which means there can only be one train occupying the track node at any time. And because of this capacity rule, the length of a track node should not be too long; otherwise the track resource can not be fully utilized. A network construction of a portion of a typical complex railway is shown in Figure 1. (Dessouky et al. 2006)

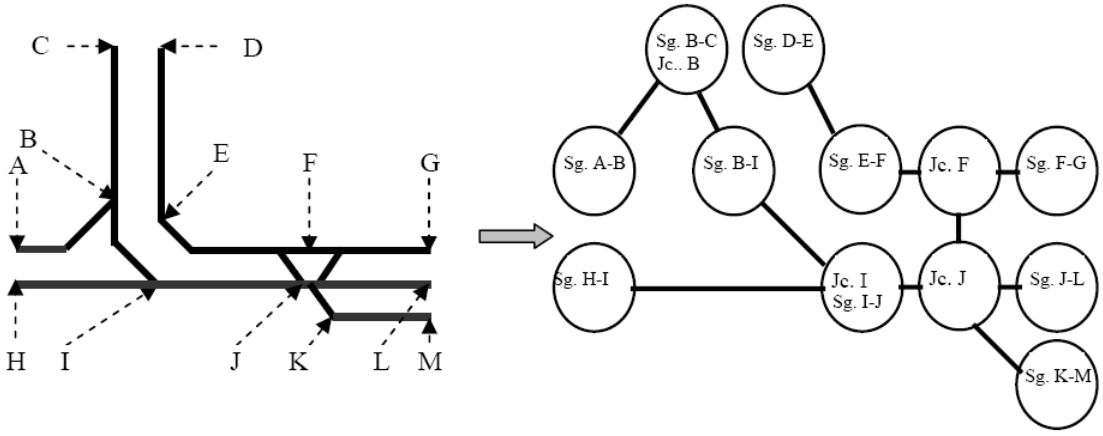

Figure 1 Network Construction

The length of the train can be longer than the length of a node. Thus a train can occupy several nodes simultaneously. In reality, a train can travel at various speeds. The acceleration and deceleration rates depend on a number of factors like locomotive power, train weight and track slope. Lu et al. (2004) and Suteewong (2006) explicitly model the acceleration and deceleration rates of the train. However, in order to make the mathematical model plausible, we assume trains travel at their maximum speed. In all the following models, no speed limits are imposed on the nodes. Trains pass each node at its maximum speed. Also the train tracks are divided into nodes with length greater than the maximum length of all the trains. This ensures a single train can

occupy at most two nodes at a time, while maintaining minimum headway clearance.

The schedule specifies the path each train takes and the arrival and departure times of each train on every node of the specified path. A path is the sequence of nodes to be traversed by the train, from its origin to its destination. Next we are going to introduce two mathematical formulations of the scheduling problem. The first formulation assumes the path for each train is given and the second formulation treats the path of each train as variables of the model.

4.2 Fixed Path Formulation

The first model in the literature that we use for benchmarking purpose is the mixed integer programming model introduced by Dessouky et al. (2006). Carey (1994b) develops a similar model which focuses on passenger railways. We refer the model formulated by Dessouky et al. (2006) as FixedPath, since the exact path of each train needs to be specified before solving the model. We now formally introduce the FixedPath model.

Notations:

Q: Set of all the trains to be scheduled

N: Set of all rail track nodes

S_q: Length of train q, $q \in Q$, $q = 1, 2, ..., |Q|$

P_q: Path train q takes. Starts with train q's origin node, n_q^0, to train q's destination node, n_q^d. All the nodes train q will be traversing are: $\{n_{q,1}, n_{q,2}, ..., n_{q,|p_q|}\}$, where $n_{q,1} = n_q^0$ and $n_{q,|P_q|} = n_q^d$

$B_{q,i}^1$: The minimal travel time between train q's head entering into node $n_{q,i}$ and train q's head leaving from node $n_{q,i}$ to node $n_{q,i+1}$

$B_{q,i}^2$: The minimal travel time between train q's head entering into node $n_{q,i}$ and train q's tail leaving node $n_{q,i}$

$t^a_{q,i}$: The time train q's head arrives at node $n_{q,i}$

$t^d_{q,i}$: The time train *q*'s tail leaves from node $n_{q,i}$

μ : Minimal safety headway between two consecutive trains

$x_{q_1,q_2,k}$: Binary variable indicates which train gets to pass node *k* first. 1: train q_1 passes node *k* before train q_2. 0: train q_2 passes node *k* before train q_1.

M: An arbitrarily large number

The 0-1 mixed integer programming formulation of FixedPath is described as follows:

$$\min \sum_{q \in Q} t^a_{q,|P_q|} \tag{1}$$

s.t.

$$t^a_{q,i+1} - t^a_{q,i} \geq B^1_{q,i}, \quad \text{for all } q \in Q \text{ and } 1 \leq i \leq |P_q| - 1 \tag{2}$$

$$t^d_{q,i} - t^a_{q,i+1} \geq B^2_{q,i} - B^1_{q,i}, \quad \text{for all } q \in Q \text{ and } 1 \leq i \leq |P_q| - 1 \tag{3}$$

$$t^d_{q,|P_q|} - t^a_{q,|P_q|} \geq B^2_{q,|P_q|}, \quad \text{for all } q \in Q \tag{4}$$

$$x_{q_1,q_2,k} M + t^a_{q_1,i} \geq t^d_{q_2,j} + \mu \quad \text{for all } q_1, q_2 \in Q \text{ and node } k = n_{q_1,i} = n_{q_2,j} \tag{5}$$

$$(1 - x_{q_1,q_2,k}) M + t^a_{q_2,j} \geq t^d_{q_1,i} + \mu \quad \text{for all } q_1, q_2 \in Q \text{ and node } k = n_{q_1,i} = n_{q_2,j} \tag{6}$$

$$x_{q_1,q_2,k} = \{0,1\} \quad \text{for all } q_1, q_2 \in Q \text{ and } 1 \leq k \leq |N| \tag{7}$$

The objective function (1) minimizes the sum of the arrival times of all trains at their destinations which is equal to the total delay of all the trains. Constraint (2) ensures the minimum traveling time of the train on each track. The equal or greater sign makes it possible for a train to wait for its next required resource to be cleared. Constraint (3) ensures the minimum time a train needs to completely clear its previous occupied resource, after its head enters the next node. The deadlock avoidance

mechanism is realized by constraints (4) and (5). These constraints together make sure that no more than one train can occupy the same node simultaneously. If train q_1 gets to pass node *k* before train q_2, the arrival time of q_2 at node *k* has to be equal to or greater than the departure time of q_1 from node *k* plus the safety headway of μ, and vice versa.

The FixedPath model can be used to solve the scheduling problem for any general network, as long as the length of each node is not shorter than the maximum length of each train. The formulation of FixedPath can be solved using a commercially available optimization solver like CPLEX. The major drawback of the FixedPath algorithm is, as its name suggests, the exact path of each train needs to be fixed and serves as the input to the model. However, the sequence of nodes a train travels is an important factor that can affect the delay of the trains. Thus the results obtained from this model are sub-optimal. To make this point clearer, suppose we have a single track network with one siding as shown in Figure 2.

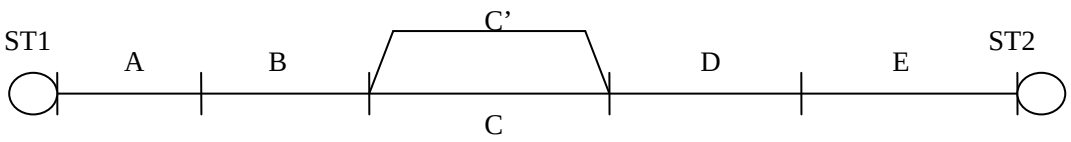

Figure 2 Simple network

Suppose trains travel in both directions, from ST1 to ST2 and from ST2 to ST1. In order to use FixedPath, for each train, we need to specify if this trains uses siding C' or not. There might be extra delays to switch from the main track to siding C'. Thus if there are no trains traveling in the opposite direction in the network, it is not optimal to dispatch the train to siding C'. On the other hand, if two trains are traveling on the network in opposite directions, one of them has to travel to the siding C' to let the other train pass. Thus the optimal path a train should take depends on the travelling direction and location of the other trains. Fixing the path before solving the scheduling problem can lead to a solution far from the global optimal solution. And if there are both slow and fast trains on the network, fixing the path might prevent the

fast train, if it follows a slower train, from overtaking the slower train.

4.3 Flexible Path Formulation

A natural extension of the FixedPath formulation is to include the path selection mechanism into the model. Here we formulate an MIP model which is extended from FixedPath, called FlexiblePath. Carey (1994b) proposes a similar model.

Let V denote the set of all junctions, where a junction is merging points of multi-tracks on the railway network. New variable I is introduced in FlexiblePath, and the meaning of variables $t^a_{q,i}$ and $t^d_{q,i}$ slightly change. Suppose there are n nodes in the network, numbered from 1 to n, respectively. $t^a_{q,i}$ ($t^d_{q,i}$) formerly indicate the arrival (departure) time of train q at (from) the i^{th} node on its path P. In FlexiblePath, $t^a_{q,i}$ ($t^d_{q,i}$) simply mean the arrival (departure) time of train q at (from) node i. Variable $I_{q,i}$ is a binary indicator variable with the following meaning:

$$I_{q,i} = \begin{cases} 1 & \text{if train } q \text{ travels on node } i \\ 0 & \text{otherwise} \end{cases}$$

Let O_q and D_q denote the origin and destination nodes of train q, respectively. Let $e(v)$ and $w(v)$, $v \in V$, denote the set of nodes connected to junction v from the East and West direction, respectively (alternatively, it can refer to the North and South directions). Let $suc(i,q)$ denote the set of nodes that are immediate successors of node i in the direction in which train q travels.

FlexiblePath formulation is as follows:

$$\min \sum_{q \in Q} t^d_{q,D_q} \tag{1}$$

s.t.

$$\left.\begin{aligned} & I_{q,O_q} = 1 \\ & I_{q,D_q} = 1 \end{aligned}\right\} \quad \text{for all } q \in Q \tag{2}$$

$$\sum_{i \in e(v)} I_{q,i} = \sum_{j \in w(v)} I_{q,j}, \quad \text{for all } q \in Q \text{ and all } v \in V \tag{3}$$

$$I_{q,i} M \geq t^a_{q,i} + t^d_{q,i}, \quad \text{for all } q \in Q \text{ and } i \in N \tag{4}$$

$$\left.\begin{aligned} & (1 - I_{q,j}) M + t^a_{q,j} - t^a_{q,i} \geq B^1_{q,i}, \\ & (1 - I_{q,i}) M + t^d_{q,i} - t^a_{q,j} \geq B^2_{q,i} - B^1_{q,i}, \end{aligned}\right\} \quad \begin{aligned} & \text{for all } q \in Q \text{ and } i \in N \\ & j \in suc(i,q) \end{aligned} \tag{5}$$

$$t^d_{q,D_q} - t^a_{q,D_q} \geq B^2_{q,D_q}, \quad \text{for all } q \in Q \tag{6}$$

$$\left.\begin{aligned} & (1 - x_{q_1,q_2,i}) M + t^a_{q_2,i} \geq t^d_{q_1,i} + \mu \\ & (2 - I_{q_1,i} - I_{q_2,i}) M + x_{q_1,q_2,i} M + t^a_{q_1,i} \geq t^d_{q_2,i} + \mu \end{aligned}\right\} \quad \begin{aligned} & \text{for all } q_1, q_2 \in Q, q_1 \neq q_2 \\ & \text{for all } i \in N \end{aligned} \tag{7}$$

$$\left.\begin{aligned} & x_{q_1,q_2,i} \leq I_{q_1,i} \\ & x_{q_1,q_2,i} \leq I_{q_2,i} \end{aligned}\right\} \quad \begin{aligned} & \text{for all } q_1, q_2 \in Q, q_1 \neq q_2 \\ & \text{for all } i \in N \end{aligned} \tag{8}$$

$$x_{q_1,q_2,i} = \{0,1\} \quad \text{for all } q_1, q_2 \in Q \text{ and } i \in N \tag{9}$$

$$I_{q,i} = \{0,1\} \quad \text{for all } q \in Q \text{ and } i \in N \tag{10}$$

Constraint (2) ensures that each train starts at its planned origin and travels to its destination. Constraint (3) is the train flow conservation equation which is also the core of the path selection mechanism. The conservation equation states that a train entering a junction can travel to any track that emits out of the junction (e.g. in Figure 2, a train can travel on either track C or C' after track B). Constraint (4) states that if a train does not utilize track node *i*, then the arrival time and departure time of that train on node *i* should be zero. Constraint (5) calculates the arrival time and departure time on track nodes along the path that the train travels. Constraint (7) is the deadlock avoidance constraint. In the formulation of FixedPath, variable $x_{q_1,q_2,i}$ only exists when both trains q_1 and q_2 have track node *i* on their paths. Here in FlexiblePath, we have the variable $x_{q_1,q_2,i}$ for any q_1 and q_2 pair on every track node. If $x_{q_1,q_2,i}$

equals to 1, this means q_1 and q_2 both travel on node *i* and q_1 gets to pass node i before q_2. If $x_{q_1,q_2,i}$ equals to 0, either one of the two situations happen: at least one of the two trains does not travel on node *i* or both q_1 and q_2 travel on node *i* and q_2 gets to pass node *i* before q_1. Constraint (8) forces $x_{q_1,q_2,i}$ to be 0 when either or both trains do not travel on node *i*.

For the same scheduling problem, the formulation of FlexiblePath contains far more binary variables than that for FixedPath. The solution computing time of FlexiblePath would be far greater than the time required for FixedPath. However a significant reduction of total delay might be achieved by incorporating the path selection mechanism of FlexiblePath. We now show the performance of both formulations by a computational experiment on a small network.

4.4 Experimental Results

The experiment is based on a portion of the real network in Los Angeles County (see Figure 3). The numbers in Figure 3 denote the lengths of the track components (in miles). The trains travelling from west to east arrive at point A and are routed to point D. For the other direction, the trains enter the network from point C and are routed to point B. From the preliminary computational experiments, we found that for this network, the maximum number of trains that FlexiblePath can solve optimally is four, given a solving time constraint of one hour of CPU time. Both formulations are tested under four scenarios. In each scenario, two trains travel in the eastbound direction and the other two trains travel in the westbound direction. The details of the four scenarios are listed in Table 1.

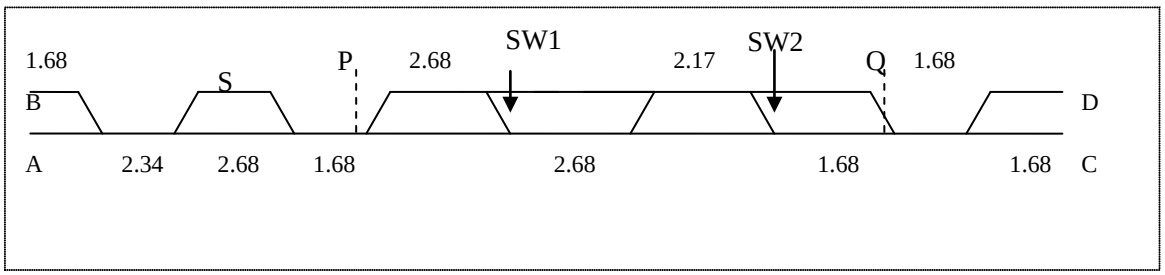

Figure 3 Small network for numerical example

Table 1 Description of the scenarios (4 trains)

4 trains	Train ready time (minute)	Train speed (miles/min)	Train length (mile)
Scenario 1	Uniform(0,10)	0.75, 1, 1.25 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 2	Uniform(0,10)	0.75 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 3	Uniform(0,20)	0.75, 1, 1.25 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 4	Uniform(0,20)	0.75 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)

The parameters of the four scenarios ensure the computational experiments mimic the real situation as close as possible. The ready times of each train is uniformly distributed. Scenarios 1 and 2 have tighter schedules than scenarios 3 and 4. The uniform distribution U(0, 10) and U(0, 20) are chosen so that there is significant difference between scenarios 1,2 and scenarios 3,4, and trains in all four scenarios are not too dense nor too sparse. In reality, the maximum speed of a passenger train can be as high as 1.35 mile/minute, whereas the maximum speed of a freight train can be as low as 0.7 mile/minute. The four possible speeds (0.75, 1, 1.25 and 1.5) ensure trains travel at different speeds as they do in reality. The average speed difference in scenarios 2 and 4 is larger than the difference in scenarios 1 and 3. Also, in reality the trains have different lengths. Typical passenger and freight trains have lengths of 0.189 and 1.136 mile. These stochastic elements lead to different meet and pass situation between trains, thus the performances of FixedPath and FlexiblePath are fully assessed. A penalty time (denoted by p) of 0.5 minute is added for each time a train switches lines (e.g. a train switches to siding from the main line). This penalty

time can be implemented by modifying constraint (5) of FlexiblePath and constraint (2) of FixedPath: if travelling from node *i* to node *j* involves switching of line, then $t^a_{q,j} - t^a_{q,i} \geq B^1_{q,i} + p$.

In order to solve the scheduling problem using the FixedPath formulation, the path of each train needs to be specified as input to the model. The path selection for each train for this numerical example is shown in Figure 4. The logic behind this assignment is to keep the trains traveling as much as possible on the right hand side whereby minimizing the number of crossings. However the disadvantage with this approach is that faster trains may possibly follow a slower train.

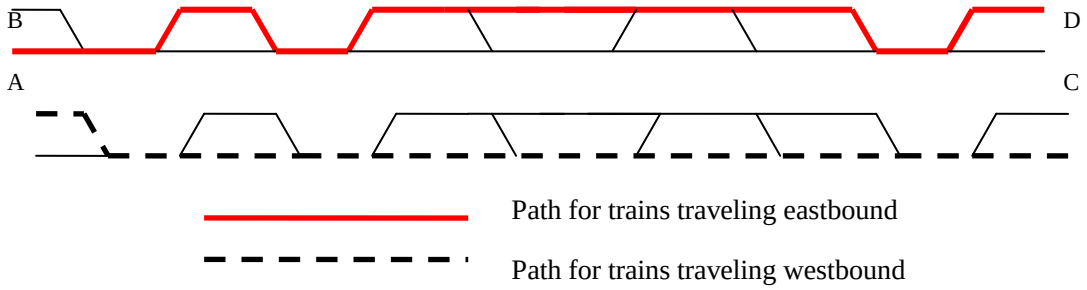

Figure 4 Path assignment for FixedPath model

20 random samples are drawn for each scenario. For each sample, both the FixedPath and FlexiblePath formulations are used to solve the scheduling problem. The experiments are conducted on a Linux server with two 3.06 GHz Intel Xeon CPUs. The software used to solve the MIP problem is CPLEX 9.0. Table 2 summarizes the computation results. The results for each scenario are obtained by averaging the 20 samples.

The schedules from FlexiblePath are optimal schedules. According to Table 2, the FlexiblePath model generates schedules with significantly fewer train delay than for FixedPath. It is observed that the reduction in scenario 2 is greater than in scenario 1, and the reduction in scenario 4 is greater than in scenario 3. One intuitive explanation is that in scenarios 2 and 4, the differences between train speeds are greater than those in scenarios 1 and 3. Since FlexiblePath is able to schedule a faster train to pass a slower train, the reduction in the delay of the faster train is greater

when there is greater difference between the speeds of the two trains. Also, if the network is less congested, there are more opportunities for a faster train to pass a slower train. Thus intuitively, the reduction in scenario 3 should be greater than the reduction in scenario 1, and the reduction in scenario 4 should be greater than scenario 2. In terms of computing time, FlexiblePath takes significantly more time than FixedPath.

Table 2 FlexiblePath V.S. FixedPath

	Total train delay under FixedPath (minute)	Total train delay under FlexiblePath (minute)	FixedPath delay /FlexiblePath delay	FixedPath CPU time (sec)	FlexiblePath CPU time (sec)
Scenario 1	12.00	7.53	1.594	0.02	256.68
Scenario 2	12.91	7.64	1.690	0.02	214.82
Scenario 3	10.50	6.48	1.620	0.02	351.94
Scenario 4	12.25	6.47	1.893	0.02	280.49

5 Algorithms for Small Networks

5.1 LtdFlePath Algorithm

Since the number of integer variables increase exponentially as the number of trains increase. Using the flexible path formulation, an optimal solution cannot be obtained within one hour of CPU time when the number of trains increases to 6 (3 in each direction). The FlexiblePath formulation tends to explore every possible path for each train. However not all paths are reasonable. For example, the path shown in Figure 5 is oddly formed, and in reality, it would never be reasonable to schedule a train on such a path. If only a subset of all possible paths is allowed to be explored, then the computing time can be reduced significantly. Still, which paths to use remains a question. We next propose a procedure to select a subset of the candidate paths. A model, similar to FlexiblePath, which only allows trains to take the candidate

paths is constructed next. The new model is called LtdFlePath.

Figure 5 An example of a poorly constructed path

There are a total of 32 possible paths (16 in each direction) in the example problem. The criteria that we use to select the candidate paths is as follows:

- The sidings are placed in the network for the purpose of meet and pass. Thus it is important to leave the siding and the main line track besides the siding as options for every train to take.
- Trains should have freedom to traverse along any track lines of the double tracks or triple tracks without switching. This minimizes delay due to crossovers. In the sample network, for the double tracks between points P and Q, trains at point P (Q) could choose the upper or lower tracks and traverse all the way towards point Q (P) without switching.
- If switching between the double or triple tracks are allowed, the first possible track to switch along the train's travel direction should be considered. By doing so, trains traveling in the same direction but at different speeds, can make use of this switch to complete the pass as early as possible. In the example, the switch denoted by SW1 can be used by the trains traveling eastbound. Thus a faster train can take over a slower train and both trains can continue traveling on the lower line after SW1, leaving the upper line for the trains traveling westbound. Under the same logic, trains traveling westbound should be allowed to make use of SW2.

Preliminary experiments show that limiting the number of paths to under six makes it possible to solve the problem using the LtdFlePath model in a reasonable amount of time. Following the three proposed criterions for selecting candidate paths, for the example network, six paths (see Figure 6) for each direction are chosen to be possible paths for model LtdFlePath. This is a reduction of the 16 possible paths that a train can take in each direction.

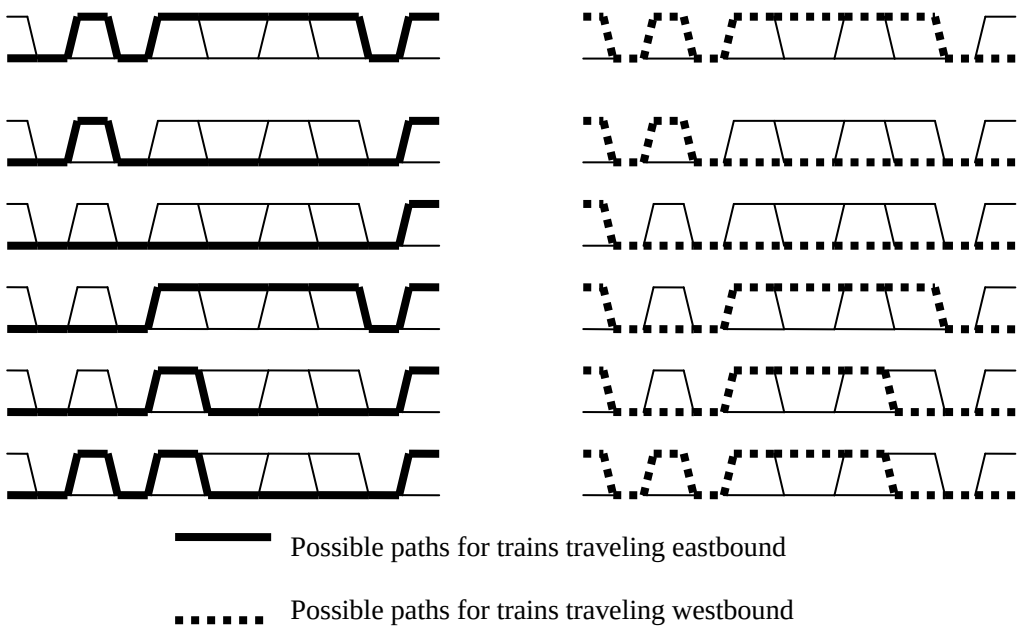

Figure 6 Candidate paths for LtdFlePath

LtdFlePath is very similar to FlexiblePath. The only difference in the formulation is in constraint (2) in FlexiblePath. LtdFlePath modifies constraint (2) so that only the candidate paths are allowed.

Table 3 FlexiblePath V.S. LtdFlePath

	Total train delay under LtdFlePath (minute)	Total train delay under FlexiblePath (minute)	LtdFlePath/FlexiblePath	LtdFlePath CPU time (sec)	FlexiblePath CPU time (sec)
Scenario 1	7.89	7.53	1.048	3.05	256.68
Scenario 2	7.82	7.64	1.024	2.53	214.82
Scenario 3	6.63	6.48	1.023	2.7	351.94
Scenario 4	6.76	6.47	1.045	2.27	280.49

Though the schedules produced by LtdFlePath are not guaranteed to be globally optimal, they are expected to be very close to the results from FlexiblePath. Using the same scenarios as before, the performance of both LtdFlePath and FlexiblePath are presented in Table 3. From Table 3, it is observed that in the case of four trains, the LtdFlePath is able to produce results that are within 5% of the global optimal

solutions. Most importantly, the LtdFlePath model significantly reduces the CPU time, as compared to FlexiblePath.

With this improved efficiency, LtdFlePath is able to solve the problem of six trains on the example network. A similar set of scenarios (see Table 4) are created to compare the performance of LtdFlePath and FixedPath. Train ready times are generated according to a uniform distribution over a larger interval than the previous scenarios. One point to note, though the total number of trains may not seem large, six trains will be running on the 18.73 miles long network simultaneously. This network scenario is actually relatively congested compared to real freight train networks.

Table 4 Description of scenarios (6 trains)

6 trains	Train ready time (minute)	Train speed (miles/min)	Train length (mile)
Scenario 1	Uniform(0,20)	0.75, 1, 1.25 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 2	Uniform(0,20)	0.75 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 3	Uniform(0,40)	0.75, 1, 1.25 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 4	Uniform(0,40)	0.75 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)

Table 5 LtdFlePath V.S. FixedPath

6 trains	Total train delay under LtdFlePath (minute)	Total train delay under FixedPath (minute)	FixedPath/LtdFlePath	LtdFlePath CPU time (sec)	FixedPath CPU time (sec)
Scenario 1	14.10	19.93	1.413	1067.98	0.11
Scenario 2	16.27	25.06	1.541	1662.18	0.09
Scenario 3	9.58	15.64	1.632	934.95	0.06
Scenario 4	10.43	19.18	1.840	266.72	0.05

The numerical results are shown in Table 5. By giving the model the freedom to optimize both the sequence of trains passing certain track segments and the sequence of tracks trains take, a much better schedule can be obtained. The LtdFlePath

formulation does not necessarily return the global optimal solution, but it is shown to perform very closely to the FlexiblePath formulation. As the results show, solutions from FixedPath can be far from optimal. However the LtdFlePath model takes much more CPU time than the FixedPath to generate a solution. Next we propose a heuristic procedure, which is improved on the FixedPath formulation. This heuristic finds a solution much faster than LtdFlePath and yet is able to significantly reduce the delay of FixedPath.

5.2 Genetic Algorithm and Fixed Path Formulation

The previous analysis showed that once the paths of the trains are specified, the scheduling problem can be solved fairly quickly (normally in the order of 0.1 CPU seconds) for the case of six trains. The heuristic, called GA+FixedPath, uses genetic algorithm to evolve the population of the candidate paths. The FixedPath model is used to calculate the fitness values for each set of paths.

The first step in solving any problem by genetic algorithm is to define the genetic representation of the population, the chromosomes. In GA+FixedPath, the chromosome represents the set of paths used by the trains. All the possible paths are first numbered accordingly. For the example problem, there are a total of 16 possible paths in each direction. They are numbered from 1 to 16 (see Figure 8 for the numbering of the paths). Instead of using 0s and 1s to represent the chromosome, the chromosome of the GA+FixedPath model are formed by the numbers that represent the selected path for each train. For the case of six trains, a chromosome might look like: (2, 3, 2, 1, 10, 1). The meaning of this chromosome is that: train 1 takes path number 2; train 2 takes path number 3; train 3 takes path number 2 and so on. Given a chromosome, the FixedPath formulation can be used to solve the scheduling problem. The returned delay is treated as the fitness value of this chromosome. Let P denote a single chromosome. The GA+FixedPath algorithm is described in the flowchart in Figure 8.

The initial population is randomly generated. However the paths do not have

equal probability of being selected by a train. The probabilities are adjusted so that a common and reasonable path has a higher probability than an odd path (e.g., path 2 in the example network should be selected with a higher probability than path 7). The probabilities of each path being selected in the initial population are shown in Figure 7. Once the initial generation is created, the FixedPath formulation is used to obtain the fitness value of each population in the initial generation. After associating each population with a fitness value, the roulette wheel selection algorithm is used to select the parent chromosomes which will be used to produce the next generation. The roulette wheel selection algorithm assures the higher the fitness a chromosome has, the higher the chance it is selected.

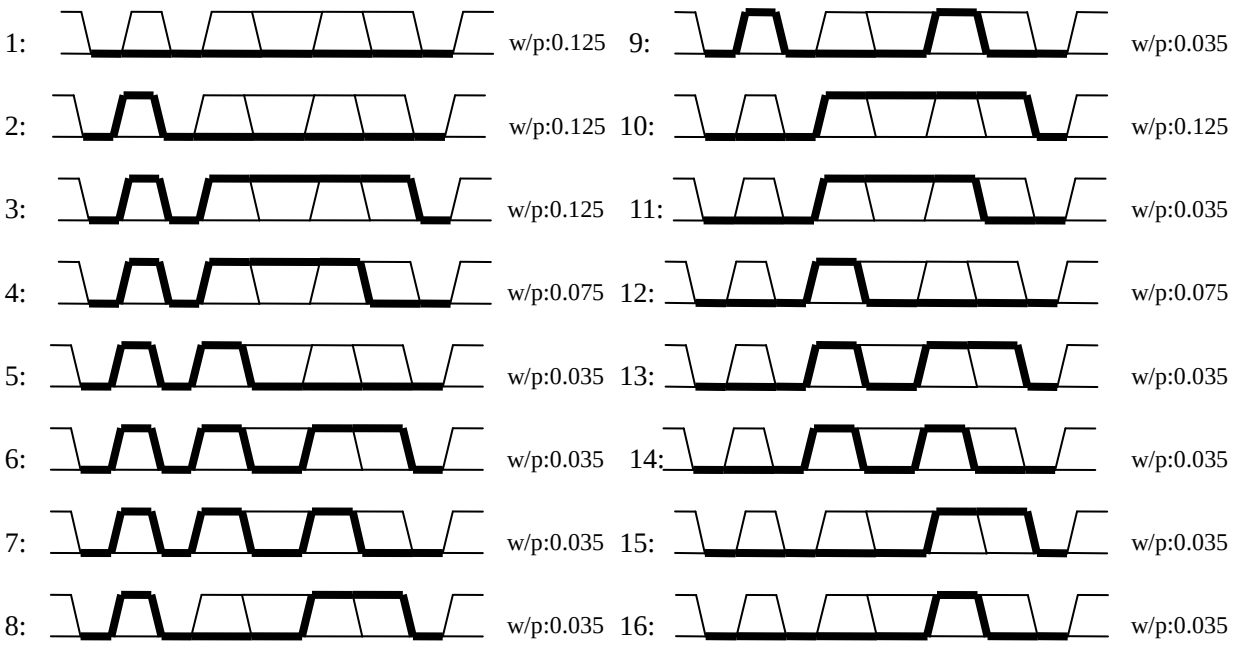

Figure 7 Numbering of the paths and their probabilities of being selected in the initial population

The crossover operation is then applied to the parent chromosomes. Since the first half of the chromosome denote the trains traveling in one direction and the second half of the chromosome denote the trains traveling in the opposite direction, it is reasonable to use the single cut point crossover policy. The single cut point is made at the middle of the chromosome. Figure 9 shows an example of the crossover operation. The crossover operation is only carried out with a certain probability which is decided by the crossover ratio.

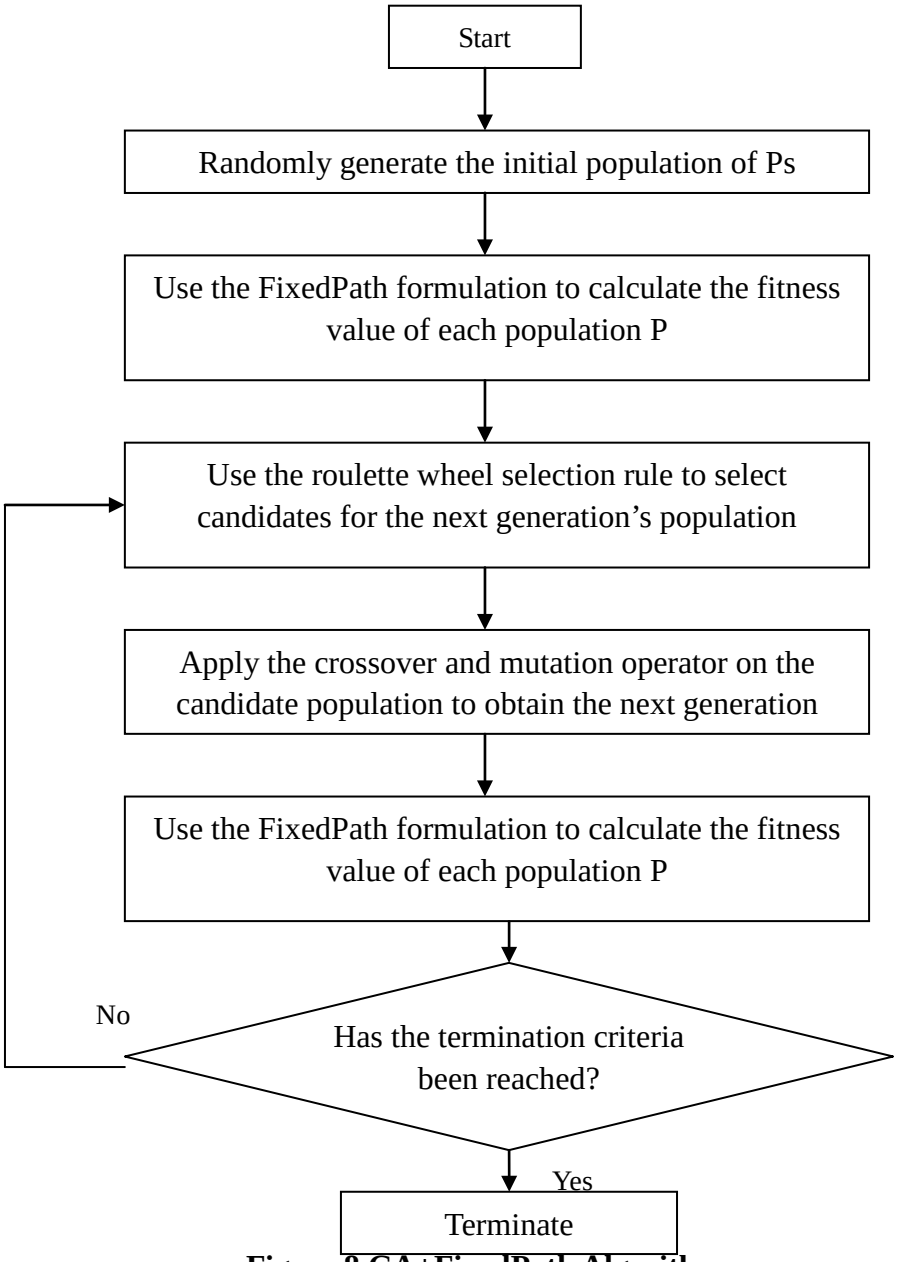

Figure 8 GA+FixedPath Algorithm

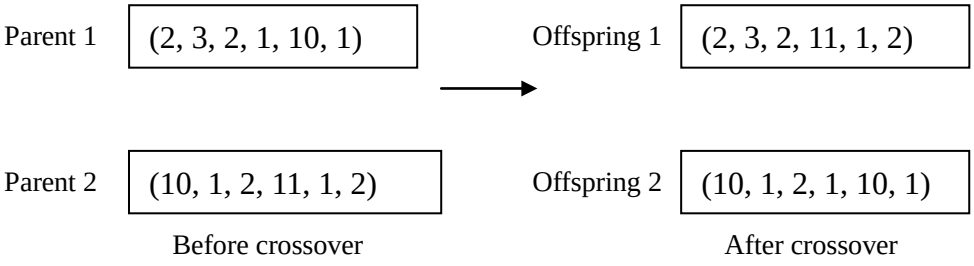

Figure 9 Example of crossover operator

After the crossover operation, the mutation operation is carried out. The mutation

operation mimics the process of human gene mutation. There is a chance (decided by a mutation ratio) that each path in a chromosome will mutate. The mutation ratio is normally set to a very low value (e.g. 0.1). A neighborhood set of paths is defined for each path. A specific path is only allowed to mutate into one of the paths in its neighborhood set. The neighborhood set of each path is composed of the paths that only slightly deviate from that path. To be more specific, the neighborhood set of path *f* contains paths that differ from path *f* by at most two switches (e.g., a neighborhood set of path 1 is path {2, 10, 11, 12, 15, 16}). In most cases, the chromosome is represented by 0s and 1s and the mutation operator normally changes 0 to 1 or vice versa. The mutations are supposed to only slightly change the current chromosome. By defining the neighborhood set for each path, this property is preserved in our algorithm. Figure 10 shows the effects of the mutation operation. The crossover operation and mutation operation are used to direct the search towards the area beyond the local optimum. The GA+FixedPath algorithm is terminated when the iteration number of the genetic algorithm reaches a pre-set value.

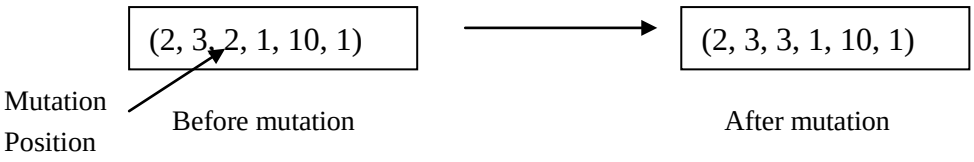

Figure 10 Example of mutation operator

The GA+FixedPath algorithm is then tested to solve the same sets of problems where six trains travel in the network. Various settings of the crossover and mutation rate are tested, the algorithm performs the best when crossover rate is 0.7 and the mutation rate is 0.1 The maximum number of iterations before termination is set to be 40. And the population size is 30. The results of GA+FixedPath are shown in Table 6. The solution quality of GA+FixedPath is much better than the one from FixedPath. The genetic algorithm is able to find a fairly good path assignment for the trains in a much shorter time compared to the computing time of LtdFlePath. The

GA+FixedPath would be able to solve the scheduling problem when the size of the problem further increases.

Table 6 LtdFlePath V.S. FixedPath V.S GA+FixedPath

	Total Delay (minutes)			Comparison		CPU time (seconds)		
6 trains	LtdFlePath	GA+FixedPath	FixedPath	GA+FixedPath/ LtdFlePath	FixedPath/ LtdFlePath	LtdFlePath	GA+FixedPath	FixedPath
Scenario 1	14.10	16.40	19.93	1.163	1.414	1067.98	93.09	0.11
Scenario 2	16.27	19.79	25.06	1.216	1.541	1662.18	78.11	0.09
Scenario 3	9.58	11.71	15.64	1.222	1.632	934.95	58.84	0.06
Scenario 4	10.43	12.99	19.18	1.246	1.840	266.72	57.02	0.05

Before we increase the size of the problem and embark our analysis on a large scale rail network, we introduce two more heuristics that schedule trains on complex rail networks. The PureGA algorithm, proposed by Suteewong (2006), schedules trains using genetic algorithm to identify a satisfactory path and priority assignment of the trains. Another heuristic is the construction heuristic proposed by Lu et al. (2004) to solve large scale scheduling problem on a complex rail network.

5.3 Pure Genetic Algorithm

Suteewong (2006) introduces a genetic algorithm to solve the train scheduling problem (referred as PureGA algorithm). While the GA+FixedPath uses genetic algorithm to evolve the paths of trains, the PureGA algorithm not only evolves the paths of each train but also the precedence rules among the trains.

In PureGA algorithm, there are two types of chromosomes. One represents the $x_{q_1,q_2,i}$ variable and the other represents the $I_{q,i}$ variable. $x_{q_1,q_2,i}$ controls the order of trains passing through a certain track node. $I_{q,i}$ equals 1 if train q passes on track i and 0 otherwise. The chromosome of $x_{q_1,q_2,i}$ is a three-dimensional matrix whose size

is $n \times n \times m$, where *n* is the total number of trains and *m* is the total number track nodes. $I_{q,i}$ has a representation that is a two-dimensional matrix whose dimension is $n \times m$. The procedure of the PureGA algorithm is as below:

1. Randomly generate an initial population for the $x_{q_1,q_2,i}$ and $I_{q,i}$.
2. Calculate the objective function for each population.
3. Use the roulette wheel selection rule to select the parent chromosomes and then use the crossover and mutation operators.
4. Obtain a new population for the $x_{q_1,q_2,i}$ and $I_{q,i}$ binary variables. Replace the previous $x_{q_1,q_2,i}$ and $I_{q,i}$ with the new ones. Evaluate the new objective function.
5. Check the termination criteria. If it is met, terminate with the solution. Otherwise, repeat step (3) through (5).

The *x* and *I* variables are correlated. As we stated in FlexiblePath, $x_{q_1,q_2,i}$ is only meaningful if $I_{q_1,i}$ and $I_{q_2,i}$ are both 1 (e.g. both trains q_1 and q_2 pass node *i*). This correlation complicates the process of evolution of the chromosomes. Before the fitness value of each population is assessed, a *Repairing Algorithm* needs to be applied to assure the consistency between *X* and *I* variables. The *Travel Time for Node Algorithm* (TTN) is developed to determine the travel time for each individual node of a particular train. Another algorithm called *Deadlock Prevention Algorithm*, which has TTN embedded in it as a subroutine, is also developed. Given the new population of the *x* and *I* variables, the *Repairing Algorithm* is first applied, and then the *Deadlock Prevention Algorithm* is applied to return a deadlock-free schedule based on the *x* and *I* variables. The PureGA algorithm, not like the GA+FixedPath algorithm, is not based on a mixed integer programming model. As we will later see, in general, the PureGA algorithm runs faster than GA+FixedPath. But in terms of solution quality, the GA+FixedPath outperforms the PureGA algorithm.

5.4 Greedy Algorithm

Lu et al. (2004) proposes a construction heuristic to schedule the trains. We call this algorithm, Greedy, since the construction heuristic is a one-step look-ahead algorithm. The Greedy algorithm is developed from a simple deadlock-free routing algorithm (call it FreePath) which allows a train to move to a successor node if all the nodes and arcs between the current position of the train and its destination are available. However, the Greedy algorithm differs from the FreePath algorithm in the way that it dispatches the train through a successor node *j* as long as there is an available buffer that passes through node *j*. A buffer between node *i* and node *j* is defined to be a set of nodes connected as a chain between node *i* and node *j*. In the simple network in Figure 11, the FreePath algorithm would hold train *A* until train *B* reaches station ST1. But the Greedy algorithm would not stop train *A* since there is a siding, which is considered as a buffer, between train *A* and *B*.

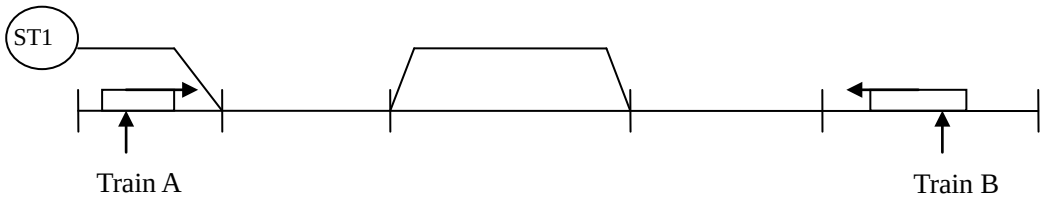

Figure 11 Greedy algorithm

When there are multiple available successor nodes for train *q*, the best available successor node is chosen according to the following factors:

1. The maximum priority difference between the current train and the immediate successor train running in the same direction if one exists.
2. The maximal number of trains running in the same direction along the path from the successor node to the train's destination node.
3. The minimum travel time for the current train from the successor node to the current train's destination node assuming there is no downstream conflicting traffic ahead of the current train.

The computing time of the Greedy algorithm is negligible even for relatively large scale problems. It is the fastest among all the algorithms introduced. And it is not surprising to see that it produces a schedule with the largest train delays.

5.5 Experimental Results

As previously pointed out, the train density of the last numerical example (six trains on an 18.73-mile long network) is considered to be rather high. Now we would like to further increase the number of the trains from six to eight. The ready times are sampled from uniform distributions from larger intervals ((0,25), (0,50)) than previous scenarios. As the train number increases, the integer variables in LtdFlePath increases exponentially. Thus makes it unable to solve the problem in a reasonable amount of time. The performances of the four algorithms are compared and the results are shown in Table 8. For the PureGA algorithm, the crossover and mutation rates are 0.6 and 0.1, respectively. The population size is set to be 50 and the maximum iteration number is set to be 100.

Table 7 Description of scenarios (8 trains)

8 trains	Train ready time (minute)	Train speed (miles/min)	Train length (mile)
Scenario 1	Uniform(0,25)	0.75, 1, 1.25 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 2	Uniform(0,25)	0.75 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 3	Uniform(0,50)	0.75, 1, 1.25 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)
Scenario 4	Uniform(0,50)	0.75 and 1.5 (equally likely)	0.189 and 1.136 (equally likely)

The GA+FixedPath algorithm requires a significant amount of computing time, compared to the other three approaches. The long computing time is justified by its solution quality. The PureGA algorithm is outperformed by the GA+FixedPath algorithm in terms of solution quality. This is because that given a specific path

assignment, the GA+FixedPath returns the optimal schedule for this path assignment. The Greedy algorithm performs the worst in terms of solution quality. The solution of the Greedy algorithm closely matches current industry practice, thus serves as a benchmarking tool which tells us how much improvement can be achieved.

Table 8 Computational results (8 trains)

	Total Delay (minutes) (CPU time (seconds))				Solution Quality Comparisons		
8 trains	FixedPath	GA+FixedPath	PureGA	Greedy	FixedPath/ GA+FixedPath	PureGA/ GA+FixedPath	Greedy/ GA+FixedPath
Scenario 1	33.97 (0.55)	30.53 (418.62)	35.22 (16.84)	78.21 (negligible)	111.25%	115.35%	256.15%
Scenario 2	56.97 (0.69)	48.64 (410.26)	56.87 (18.75)	83.25 (negligible)	117.12%	116.92%	171.14%
Scenario 3	35.19 (0.16)	26.29 (215.25)	30.66 (11.06)	72.32 (negligible)	133.85%	116.59%	275.04%
Scenario 4	42.61 (0.23)	34.79 (293.67)	40.38 (12.98)	65.79 (negligible)	122.46%	116.05%	189.09%

The PureGA and Greedy algorithm are also tested for the scenarios of four and six trains. The complete tables which have all the results are shown in Appendix A. The PureGA tends to work well when the number of trains is small. As the number of trains increases, the solution quality of PureGA gradually approaches to that of the FixedPath algorithm. Even when the number of trains is small, there is a significant gap in terms of solution quality between the Greedy Algorithm and the optimal algorithms. This suggests that the freight rail industry can benefit a lot if better and practical operational scheduling methods are implemented.

6 Algorithms for Large Networks

In Section 4, we developed and tested a few algorithms on actual small rail networks. The optimal schedule (obtained from FlexiblePath) is benchmarked with the schedule practiced in industry (obtained from Greedy) and some other sub-optimal schedules. The solution gap is clearly identified. Now we embark on solving large scale problems. For these large scale problems, the global optimal solution is not computationally practical to obtain. Any heuristic proposed to solve the large scale problem can be tested using the small network so that its relative performance compared to the optimal solution can be inferred.

Among all the algorithms introduced in Section 4, only the Greedy algorithm can solve a large scale problem in a reasonable amount of time. In this section, we propose a more efficient heuristic than the Greedy Algorithm to solve the scheduling problem for a large scale real network.

6.1 Decomp Algorithm

The divide-and-conquer approach is an effective way of solving large complex problems. For a static large scale operational scheduling problem, there can be two ways to decompose the problem.

- Horizontal decomposition. The rail network can be decomposed into several smaller sections. Local optimal schedules could be developed and then schedules on each smaller section are then integrated. The major challenge of horizontal decomposition is the integration step (e.g., the leaving time of a train in one section should match the arrival time of this train to the next adjacent section).
- Vertical decomposition. Instead of decomposing the large network into smaller sections, the trains to schedule can be grouped into clusters, according to the time the trains enter the network. Then a schedule can be developed for each cluster, assuming the network is occupied by the trains only from the current cluster. And the schedules of each cluster are then integrated.

Carey and Lockwood (1995) solve the train scheduling problem by dispatching the trains one by one, similar to the procedure carried out by a human dispatcher. This approach can be categorized as vertical decomposition. Next, we are going to introduce an algorithm, called Decomp algorithm, which is also based on the idea of vertical decomposition.

The train scheduling problem is basically a problem of determining the paths of each train on their routes (the *I* variables in the FlexiblePath formulation) and the sequence of the trains passing every track segment (the *x* variables in the FlexiblePath formulation). Before introducing the Decomp algorithm, some additional notation needs to be defined:

tr_k : The *k*th train entering the network, $1 \le k \le |Q|$

c: The total number of clusters

C_l : The *l*th cluster of trains, $1 \le l \le c$

$|C_l|$: The total number of trains in C_l

s_l : The total number of trains up to cluster *l*. $s_l = \sum_{m=1}^{l} |C_m|$

<u>Decomp Algorithm</u>

Step 1: Decompose all the trains into clusters according to the entering time of the trains. C_l will contain trains $tr_{s_{l-1}+1}$, $tr_{s_{l-1}+2}$ up to tr_{s_l} .

Step 2:

Let *l* = 1;

While (*l* < *c*+1) {

Solve the scheduling problem (referred as sub-problem *l*) that only involves trains in clusters $C_1, C_2, ..., C_h$;

Save and fix the values of variables $I_{q,i}$ and $x_{q_1,q_2,i}$, where $q, q_1, q_2 \in \cup(C_1, C_2, ..., C_h)$ and $i \in N$, for the next iteration;

l= *l* +1;

}

The description of the Decomp algorithm only serves as a general approach, there are two details that need to be decided.

1. The cluster size. The trade off between a large and small cluster size is very clear. The larger the cluster size is, the better the solution quality is. But the larger the cluster size is, the longer it takes to solve the sub-problem.
2. Algorithm used to solve sub-problems. Since the sub-problem itself is a smaller version of the train scheduling problem, there are many options in solving the problem. As the cluster size varies, the most suitable algorithm will change.

After solving sub-problem *l*, the path of each train in clusters $C_1, C_2 ... C_h$ will be fixed, so will the sequence for those trains passing each track node. Solving sub-problem *l* involves determining the paths for the trains in cluster C_h, the sequence of the trains in C_h passing every node, and the precedence relationship between trains in C_h and trains in $C_1, C_2, ..., C_{h-1}$. Thus if every cluster has the same size, the problem size of the sub-problems continuously increases (e.g., it takes more time to solve sub-problem *l*+1 than sub-problem *l*).

6.2 Experimental Results

Next we demonstrate the performance of the Decomp algorithm by testing it on a real network, to be more specific, a 49.3-mile long network from Indio, CA to Colton, CA. The trackage configuration of this network is shown in Appendix B. Similar to before, we gradually increase the number of trains and explore the performance of the various algorithms. The time interval between two consecutive trains in the same direction is assumed to be uniformly distributed between 14 and 16 minutes. The train speed is equally likely to be 0.75, 1, 1.25 and 1.5 miles/minute and the length of each train is equally likely to be 0.189 and 1.136 miles.

For this particular numerical example, the cluster size is chosen to be fixed at six (except for the last cluster). To solve the sub-problem, the GA+FixedPath algorithm

(population size: 40; crossover rate: 0.7; mutation rate: 0.1; maximum number of iterations: 15) is found to be most suitable. The PureGA and Greedy algorithms are the only two algorithms introduced that could, in a reasonable time duration, solve this particular example without modifications. Thus the results from the Decomp algorithm are benchmarked against PureGA and Greedy algorithm. The results based on 20 random samples for each scenario are shown in Table 9.

Table 9 Computational results (large network)

	Total Delay (minutes)			Comparison		CPU time (seconds)		
	Decomp	PureGA	Greedy	PureGA/Decomp	Greedy/Decomp	Decomp	PureGA	Greedy
8 trains	33.31	44.60	51.35	1.339	1.543	60.7	248.5	negligible
10 trains	48.44	52.12	80.27	1.282	1.662	217.9	587.7	negligible
12 trains	52.41	83.36	101.29	1.590	1.933	298.0	1351.9	negligible
14 trains	66.92	101.14	117.45	1.511	1.755	497.6	1540.0	negligible
16 trains	74.96	110.55	134.88	1.475	1.799	716.6	2057.0	negligible
18 trains	94.23	140.77	154.54	1.494	1.640	1155.5	2544.5	negligible
20 trains	87.40	137.64	153.71	1.575	1.759	1518.5	2634.0	negligible
22 trains	100.55	170.33	182.13	1.694	1.811	1992.0	2818.0	negligible
24 trains	112.28	173.84	191.18	1.548	1.703	4658.9	3271.0	negligible

The relative performances between PureGA and Greedy are consistent with their performances in the sample network. The Decomp algorithm achieves a much smaller delay than the PureGA and Greedy algorithms. For the Decomp algorithm, though the problem is decomposed into smaller problems, the sizes of the sub-problems are not constant. For the case of 18 trains, the average CPU times to solve sub-problems 1, 2 and 3 are 43.38, 368.69 and 743.25 seconds, respectively. When the number of trains increases, it might require too much time to solve the last few sub-problems. The

Decomp algorithm might need to be modified to be less sensitive to the size of the problem.

We then show the sensitivity analysis of the cluster size. Table 10 shows the effects of different cluster sizes in terms of solution quality and CPU times. For the case of 16 trains, we can decompose trains in three ways: (1) cluster size of 4: (4, 4, 4, 4); (2) cluster size of 6: (6, 6, 4); (3) cluster size of 8: (8, 8). The result shows that the larger the cluster size is, the better the solution quality and the longer the solution time are. For a balance between solution quality and solution time, for the previous numerical examples, the cluster size is chosen to be fixed at six (e.g., for the case of 22 trains, the decomposition will be 6, 6, 6 and 4).

Table 10 Effects of different cluster size (16 trains)

Total Delay (minutes)			CPU time (seconds)		
Cluster size of 4	Cluster size of 6	Cluster size of 8	Cluster size of 4	Cluster size of 6	Cluster size of 8
77.06	74.96	73.3	368.23	716.65	5640.57

6.3 Parallel Algorithm

Now we present another algorithm, called Parallel, which is also based on the decomposition idea. The Parallel algorithm is designed to be less sensitive to the size of the problem.

Parallel algorithm

Step 1: Decompose all the trains into clusters according to the entering time of the trains. C_l will contain trains $tr_{s_{l-1}+1}$, $tr_{s_{l-1}+2}$ up to tr_{s_l}.

Step 2:

Let $l = 1$;

While ($h \le c$) {

Solve the scheduling problem (referred as sub-problem *l*) that only involves trains in clusters C_h;

Save and fix the values of variables $I_{q,i}$ and $x_{q_1,q_2,i}$, where $q, q_1, q_2 \in C_h$ and $i \in N$;

l= *l* +1;

}

Step 3:

Use FixedPath formulation to solve the problem with all Q trains. The path of each train are fixed as in the solution of Step 2. Some of the *x* variables are also fixed as follows:

1. For trains q_1 and q_2 belonging in the same cluster, $x_{q_1,q_2,i}$ is fixed as in the solution in Step 2, $i \in N$
2. If $I_{q_1,i} = I_{q_2,i} = 1$, set $x_{q_1,q_2,i} = 1$ and $x_{q_2,q_1,i} = 0$, where $q_1 \in C_l$ and $q_2 \in C_s$, $s \geq l + 2$, for $l = 1..c-2$ and $s = 3..c$.

The main distinction between Step 2 of the Parallel algorithm and the Decomp algorithm is that the Parallel algorithm involves sub-problems that are independent of each other (i.e., when determining the paths of trains in cluster C_2, the Parallel algorithm does not consider the paths of trains in C_1, whereas the Decomp algorithm considers them). The major drawback in terms of the computation time for the Decomp algorithm is that the sizes of the sub-problems continuously increase. The Parallel algorithm does not have this drawback; every sub-problem is of the same size, if the cluster sizes are the same.

The algorithm is called Parallel, because in Step 2, a total number of *c* sub-problems are solved independently, thus all the sub-problems can be solved in parallel. Nowadays, most CPUs in personal computers have multiple processing cores. By solving the problem in parallel, the computation time of Step 2 can be reduced by a factor of the number of CPU cores.

Step 2 determines the paths of each train. Step 3 involves solving a scheduling

problem where the path of each train is given. However, given a relatively large network and large number of trains, the FixedPath formulation can not solve Step 3 efficiently without pre-fixing some of the *x* variables. The precedence rules for trains in the same cluster are pre-fixed as the solution in Step 2. Also, to further reduce the solution space, the trains in cluster *l* have precedence on every track node over all the trains in clusters *l+2, l+3, …, c*. So, in Step 3, the FixedPath formulation is used to only determine the precedence rule between trains in adjacent clusters.

6.4 Experimental Results

For the example large network, the cluster size is set to be 10 for the Parallel algorithm. This cluster size is bigger than the one used in Decomp algorithm. The reason being that the size of the sub-problem is constant for the Parallel algorithm, using a bigger cluster will not result in intractable sub-problems. We want the cluster size to be as big as possible while keeping the sub-problems solvable in reasonable time duration, thus a cluster size of 10 is used. For the sub-problems in Step 2, the GA+FixedPath algorithm is found to be most suitable. The server we used to conduct our experiments has two CPU cores. Thus the sub-problems can be solved in parallel, two problems at a time.

We first show the comparison between the Decomp algorithm and Parallel algorithm, in terms of both solution quality and solution time. Table 11 summarizes the results for the scenarios of 20 trains. Other experiment parameters are the same as in Section 5.2. As expected, the solution quality of the Parallel algorithm is worse than the one for the Decomp algorithm, since the sub-problems are solved independently. However, the solution time of Parallel algorithm is less. And when the number of trains increases, the gap between the solution times becomes significant.

Table 11 Comparison between Decomp and Parallel algorithm (20 trains)

Total Delay (minutes)		CPU Time (seconds)	
Decomp	Parallel	Decomp	Parallel
87.40	101.09	1518.50	596.6

Table 12 shows the experimental results of the Parallel algorithm for scenarios of 20, 30 and 40 trains. For 30 and 40 trains, neither the Decomp nor PureGA algorithm could solve the problem in a reasonable amount of time. While being able to return a better solution than the Greedy algorithm, the solution times of the Parallel algorithm do not increase as rapidly as for the Decomp algorithm.

Table 12 Computational results (large network)

	Total Delay (minutes)		Comparison	CPU Time (seconds)	
	Parallel	Greedy	Greedy/Parallel	Parallel	Greedy
20 trains	101.09	153.71	1.52	596.6	Negligible
30 trains	182.29	251.79	1.38	1782.41	negligible
40 trains	279.60	333.81	1.19	2411.77	negligible

7 Dynamic Scheduling

7.1 Dynamic Algorithm

So far, all the problems discussed are for the static scheduling problem. For the static scheduling problem, the arrival time information for all trains is known before solving the problem. However, in reality, at the time the first few trains enter the network, the information about the arrival time of the later trains may not be known in advance. In dynamic scheduling, the information of only arrived trains is considered known. Then the schedule of the new train and the trains currently in the network

should be generated, given no information of later trains.

In the environment of dynamic scheduling, the time constraint to generate each schedule is very tight. From the moment the information of the new arriving train is known, to the moment the schedule of the new train needs to be executed, the time constraint can be rather tight. Given this tight time constraint, it is normally not possible to re-optimize the whole schedule of all the trains in the network. Next, we present an algorithm, called Dynamic, which uses the idea of vertical decomposition and sequential optimization.

When the new train enters the network, the paths of the existing trains and precedence rules among those trains are all determined by the previous schedules. The fastest way to generate a schedule is to fix the paths and precedence rules for all the existing trains and only generate the path for the new train and the precedence rule between this new train and the existing trains. Given no information about later trains, when a new train enters the network, the best schedule is obtained by optimizing the paths and precedence rules for all trains, given the current location of the existing trains in the network. Re-optimizing the paths and precedence rules for all the trains in the network gives the best schedule with regard to all the current trains but doing so takes too much computational time. On the other hand, optimizing the path and precedence rule for only the new train requires the least time, but the solution quality is worse. If we fix the paths and precedence rules for some of the existing trains, and optimize the paths and precedence rules for the rest of the existing trains, altogether with the new train, we may achieve a better balance between the solution quality and solution time.

Dynamic Algorithm

Step 1: Record the locations of the existing trains in the network as the new train enters the network.

Step 2: Determine which trains of the existing trains are free to be re-scheduled.

Step 3: Fix the paths and precedence rules of the existing trains, which are not to be re-scheduled, as previously determined.

Step 4: Optimize the paths and the precedence rules of the existing trains which are to

be re-scheduled, together with the newly entered train.

There are two details of the algorithm that need further consideration. In Step 2 of the algorithm, we need to determine how many and which of the existing trains are to be re-scheduled. The more trains to be re-scheduled, the better the solution quality and the longer the solution time is. And in Step 4, we have a standard train scheduling problem. We need to determine which algorithm to use to solve the train scheduling problem.

7.2 Experimental Results – Small Networks

We first demonstrate the performance of the Dynamic algorithm by testing it on a small network. The small network illustrated in Figure 3 is used. The train speed is equally likely to be 0.75, 1, 1.25 and 1.5 miles/minute and the length of each train is equally likely to be 0.189 and 1.136 miles. The arrivals of the trains in each direction follow a Poisson Process with an inter-arrival time of 9 minutes.

For Step 2 of the algorithm, we are going to test two scenarios. In scenarios 1, we fix the paths and precedence rules of all existing trains. Thus the train scheduling problem in Step 4 only determines the path of the newly entered train and the precedence rule between the newly entered train and all existing trains. In scenarios 2, we let one of the existing trains be re-scheduled. The existing train which is closest to the entering point of the new train is chosen to be re-scheduled. Intuitively, the train closest to the new train might have great impact on the path of the new train, thus by making the path of the closest train flexible, a better overall schedule might be generated. For Step 4 of the algorithm, the FlexiblePath formulation is used to solve the train scheduling problem.

Table 13 Computational results of Dynamic algorithm (small network)

	Average Delay Per Train (minutes)			Average CPU Time Per Iteration (seconds)		
Total trains	Dynamic-1	Dynamic-2	Greedy	Dynamic-1	Dynamic-2	Greedy
20	5.64	4.35	11.31	0.08	0.81	negligible
40	7.27	4.85	16.51	0.11	0.87	negligible
60	6.78	4.53	18.02	0.11	0.83	negligible
80	8.37	6.36	21.71	0.13	0.86	negligible
100	8.46	6.30	19.30	0.15	2.01	negligible

Table 13 shows the computational results of the Dynamic algorithm. We compare the Dynamic algorithm with the Greedy algorithm. The Greedy algorithm is a one-step look-ahead heuristic, thus the Greedy algorithm can solve the dynamic scheduling problem. In Table 13, Dynamic-1 refers to the scenario where paths and precedence rules of all existing trains are fixed, whereas, Dynamic-2 refers to the scenario where the path and precedence rule of one of the existing trains can be optimized, together with the newly entered train. Both Dynamic-1 and Dynamic-2 generate schedules with much smaller train delays than the schedules generated by the Greedy algorithm. The schedules from the Dynamic-2 algorithm achieve lower delays than the schedules from the Dynamic-1 algorithm, as expected. Every time a new train enters the network, we need to solve a scheduling problem which schedules the new train and the existing trains. The CPU times in Table 13 denote the average time taken to solve the scheduling problem every time a new train enters. Since in Dynamic-2, we have a bigger scheduling problem to solve in Step 4 of the algorithm, the increase in the CPU time of Dynamic-2, as compared to Dynamic-1, is expected. And the CPU time exponentially increases, as we make more existing trains flexible to be re-scheduled. We can conclude that the Dynamic algorithm works very well on small networks; it takes very little time to dynamically generate good schedules.

7.3 Experimental Results – Large Networks

Now we test the performance of the Dynamic Algorithm on a relatively large network. The same network as in Section 5 is used. The network is about 49.3 miles long. Trains are traveling both eastbound and westbound. The arrival pattern of new trains in each direction follows a Poisson Process with an inter-arrival time of 15 minutes. As before, the train speed is equally likely to be 0.75, 1, 1.25 and 1.5 miles/minute and the length of each train is equally likely to be 0.189 and 1.136 miles.

The large network has significantly more nodes than the small networks, thus we expect the solution time of the Dynamic algorithm to increase exponentially, as compared to the case of small networks. To the reduce problem size of the train scheduling problem in Step 4 of the algorithm, instead of using the FlexiblePath formulation to solve the train scheduling problem, the LtdFlePath algorithm is used to solve the train scheduling problem. Preliminary results show that, for this relatively large network, the Dynamic-2 algorithm, which frees one of the existing trains to be re-scheduled, cannot generate a schedule in a short time constraint that is required by the dynamic environment. Thus we only compare the performance of the Dynamic-1 algorithm with the Greedy algorithm. The results are shown in Table 14.

Table 14 Computational results of Dynamic algorithm (large network)

	Average Delay Per Train (minutes)		Average CPU Time Per Iteration (seconds)	
Total trains	Dynamic-1	Greedy	Dynamic-1	Greedy
20	4.82	8.86	4.1	negligible
40	5.58	10.89	6.87	negligible
60	5.39	8.85	4.99	negligible
80	5.45	10.57	5.89	negligible
100	6.06	9.87	8.83	negligible

Like for small networks, the Dynamic algorithm outperforms the Greedy algorithm for large networks. The average solution time when a new train enters the network is well under 10 seconds. Thus the Dynamic algorithm can be applied in real time.

8 Implementation

This project addresses the area of Commercial Goods Movement and International Trade. Freight train transportation is a cost effective way to move goods from ports to inland destinations. According to the Association of American Railroads, more than 40% of all freight is transported by trains in the US. Given the fact that the freight railroad industry is already running without much excess capacity, better planning and scheduling tools are needed for railroad management. In particular, this research focuses on solving the freight train scheduling and dispatching problems.

The optimization based scheduling heuristics developed in this research are tested on real-world rail networks in the Los Angeles area. The performance of the proposed heuristics is compared with the performance of existing heuristics in the literature. The heuristics developed outperforms the existing heuristics. The implementation of our heuristics will require suitable optimization software tools such as CPLEX, and access to railway data such as train lengths, train speeds, headway regulations and ready times of trains.

9 Conclusion

According to a study conducted by the Association of American Railroads, trains move about 40% of all freight in the US. And the demand for rail transportation will increase rapidly in the near future. Given the fact that the freight railway industry is already running without much excess capacity, better planning and scheduling tools are needed to effectively manage the scarce resources, in order to cope with the

rapidly increasing demand for railway transportation. Train scheduling and dispatching is one important sub-problem of the freight railroad management problem. In this report, we propose heuristics for both the static and dynamic scheduling of freight trains. In the literature of static scheduling, most of the research simplifies the rail network; our heuristics can work on any complex rail network. We first introduce exact methods for solving the static train scheduling problem. Then we present few heuristics which can significantly reduce the solution time, yet produce a satisfactory solution quality. We also compare our heuristics with three existing procedures. Our heuristics are able to produce better solutions in terms of minimizing delay, in a reasonable amount of time. For static scheduling in large networks, two heuristics based on the idea of decomposition are proposed. Both algorithms significantly outperform existing algorithms. Then we move to dynamic scheduling of freight trains. While the literature on dynamic scheduling is very limited, a heuristic based on sequential optimization is proposed. Experimental results show that the Dynamic algorithm is able to reduce delay by at least 40% of existing algorithm on representative rail scenarios. For future work, we plan to use techniques like queuing theory to analyze the delay structure of some typical simple track configurations. The results of the theoretic analysis might serve as the guidance for smarter greedy heuristics.

References

Adenso-Diaz, B., Oliva Gonzalez, M., & Gonzalez-Torre, P. (1999). On-line timetable rescheduling in regional train services, *Transportation Research Part B*, 33, 378–398.

Ahuja, R. K., Cunha, C. B., & Sahin, G. (2005). Network models in railroad planning and scheduling. In H. J. Greenberg & J. C. Smith (Eds.), *TutORials in Operations Research*, 54–101

Brown, D.E., Huntley C.L., Markowicz, B.P., & Sappington D.E. (1992). Rail network routing and scheduling using simulated annealing, in *Proceedings of the 1992 IEEE International Conferenceon Systems, Man, and Cybernetics*, 1, 589–592.

Brannlund, U., Lindberg, P. O., Nou, A., & Nilsson, J. E. (1998). Railway timetabling using lagrangian relaxation, *Transportation Science*, 32, 358–369.

Caprara, A., Kroon, L. G., Monaci, M., Peeters, M., & Toth, P. (2006). Passenger railway optimization. In C. Barnhart & G. Laporte (Eds.), *Handbooks in Operations Research and Management Science*, 14, 129–187.

Caprara, A., Fischetti, M., & Toth, P. (2002). Modeling and solving the train timetabling problem, *Operations Research*, 50, 851–861.

Carey, M. (1994a). A model and strategy for train pathing with choice of lines, platforms, and routes, *Transportation Research Part B*, 28 (5), 333-353.

Carey, M. (1994b). Extending a train pathing model from one-way to two-way track, *Transportation Research Part B*, 28 (5), 395-400.

Carey M. & Lockwood, D. (1995). A model, algorithms and strategy for train pathing, *Journal of Operation Research Society*, 46, 988-1005.

Cordeau, J., Toth, P. & Vigo, D. (1998). A survey of optimization models for train routing and scheduling, *Transporation Science*, 32 (4), 380-404.

Crainic, T. G. (2003). Long-haul freight transportation. In R. W. Hall (Ed.), *Handbooks in Transportation Science*, 56, 451–516.

D'Ariano, A., Pacciarelli, D., & Pranzo, M. (2007). A branch and bound algorithm for scheduling trains in a railway network, *European Journal of Operational Research*, 183 (2), 643–657.

D'Ariano, A., (2008). Improving real-time train dispatching: models, algorithms and applications, Ph.D. Thesis, Technische Universiteit Delft.

Dessouky, M.M. & Leachman, R.C. (1995). A simulation modeling methodology for analyzing

large complex rail networks, *Simulation*, 65, 131-142

Dessouky M.M., Lu, Q. & Leachman, R.C. (2002). Using simulation modeling to assess rail track infrastructure in densely trafficked metropolitan areas, in *Proceedings of the 2002 Winter Simulation Cconference*, 725-231.

Dessouky M., Lu, Q., Zhao, J. & Leachman, R.C. (2006). An exact solution procedure for determining the optimal dispatching times for complex rail networks, *IIE Transaction*, 28, 141-152.

Dorfman, M.J. & Medanic, J. (2004). Scheduling trains on a railway network using a discrete event model of railway traffic, *Transportation Research Part B*, 38, 81-98.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to Theory of NP-Completeness, New York: Freeman.*

Ghoseiri, K., Szidarovszky, F. & Asgharpour, M. J. (2004). A multi-objective train scheduling model and solution, *Transportation Research Part B*, 38, 927–952.

Higgins, A., & Kozen, E. (1997). Heuristic techniques for single line train scheduling, *Journal of Heuristics*, 3, 43-62.

Higgins, A., Kozan, E., & Ferreira, L. (1996). Optimal scheduling of trains on a single line track, *Transportation Research Part B*, 30, 147–161.

Huntley C.L., Brown, D.E., Sappington D.E., & Markowicz, B.P. (1995). Freight routing and scheduling at CSX transportation, *Interfaces*, 25 (3), 58-71.

Jovanovic, D., & Harker, P. T. (1990). A decision support system for train dispatching: an optimization-based methodology, *Journal of the Transportation Research Forum*, 31, 25–37.

Jovanovic, D., & Harker, P. T. (1991). Tactical scheduling of train operations: The SCANI system, *Transportation Science*, 25, 46–64.

Jacobs, J. (2004). Reducing delays by means of computer-aided ‘on-the-spot’ rescheduling. In J. Allan, C. A. Brebbia, R. J. Hill, G. Sciutto, & S. Sone (Eds.), *Computers in Railways IX, 603–612. Southampton, UK: WIT Press.*

Kraay, D. R., & Harker, P. T. (1995). Real-time scheduling of freight railroads, *Transportation Research Part B*, 29, 213–229.

Kraay, D. R., Harker, P. T., & Chen, B. (1991). Optimal pacing of trains in freight railroads: model formulation and solution, *Operations Research*, 39, 82–99.

Leachman, R. C. (2002) *Los Angeles – inland empire railroad main line advanced planning study,* prepared for the Southern California Association of Governments, contract number 01-077, work element number 014302, October 1, 2002.

Lu, Q., Dessouky M. & Leachman, R.C. (2004). Modeling train movements through complex rail networks, *ACM Transactions on Modeling and Computer Simulation,* 14 (1), 48-75.

Mascis, A., Pacciarelli, D., & Pranzo, M. (2001, June). Train scheduling in a regional railway network, in *Preprints of the 4th Triennial Symposium on Transportation Analysis*, 487–492. Sao Miguel, Portugal.

Nachtigall, K. & Voget, S. (1996). A genetic algorithm approach to periodic railway synchromization, *Computer and Operation Research,* 23 (5), 453-463.

Nachtigall, K. & Voget, S. (1997). Minimizing waiting times in integrated fixed interval timetables by upgrading railway tracks, *European Journal of Operational Research*, 103, 610-627.

Ping, L., Axin, N., Limin, J. & Fuzhang, W. (2001). Study on intelligent train dispatching, in *Proceedings of 2001 Intelligent Transportation Systems Conference*, 949-953.

Rodriguez, J. (2007a). A constraint programming model for real-time train scheduling at junctions, *Transportation Research Part B*, 41 (2), 231–245.

Ruan, W., Giras, T.C., Lin, Z. & Ou, Y. (2003). ASCAP parameter determination by an intelligent genetic algorithm, in *Proceedings of the 2003 IEEE/ASME Joint Rail Conference*, 133-141.

Sahin, I. (1999). Railway traffic control and train scheduling based on inter-train conflict management, *Transportation Research Part B*, 33, 511–534.

Salim, V. & Cai, X. (1995). Scheduling cargo trains using genetic algorithms, in *ICEC'95*, 224-227.

Szpigel, B. (1973). Optimal train scheduling on a single track railway. In M. Ross (Ed.), *Operational Research '*72, 343–352. Amsterdam, the Netherlands.

Tornquist, J., & Persson, J. A. (2007). N-tracked railway traffic re-scheduling during disturbances, *Transportation Research Part B*, 41 (3), 342–362.

Wegele, S. & Schnieder, E. (2004). Dispatching of train operations using genetic algorithms, *CASPT conference paper*.

Zhou, X., & Zhong, M. (2005). Bicriteria train scheduling for high-speed passenger railroad planning applications, *European Journal of Operational Research*, 167, 752–771.

Zhou, X., & Zhong, M. (2007). Single-track train timetabling with guaranteed optimality: branch-and-bound algorithms with enhanced lower bounds, *Transportation Research Part B*, 41, 320–341.

Appendix A: Complete set of results for sample network

Four trains case:

4 trains	Total Delay (minutes) (CPU time (seconds))					
	FlexiblePath	LtdFlePath	GA+FixedPath	PureGA	FixedPath	Greedy
Scenario 1	7.53 (256.68)	7.89 (3.05)	9.11 (17.08)	9.84 (0.86)	12.00 (0.02)	22.09 (negligible)
Scenario 2	7.64 (214.82)	7.82 (2.53)	8.78 (13.21)	10.20 (0.95)	12.91 (0.02)	28.75 (negligible)
Scenario 3	6.18 (351.94)	6.63 (2.70)	7.73 (13.79)	8.19 (0.95)	10.30 (0.02)	19.57 (negligible)
Scenario 4	6.47 (280.49)	6.76 (2.27)	8.15 (13.10)	8.53 (0.73)	12.25 (0.02)	24.31 (negligible)

Six trains case:

6 trains	Total Delay (minutes) (CPU time (seconds))				
	LtdFlePath	GA+FixedPath	PureGA	FixedPath	Greedy
Scenario 1	14.10 (1067.98)	16.40 (93.09)	19.20 (5.52)	19.93 (0.11)	47.51 (negligible)
Scenario 2	16.27 (1662.18)	19.79 (78.11)	23.06 (5.58)	25.06 (0.09)	53.51 (negligible)
Scenario 3	9.58 (934.95)	11.71 (58.84)	13.96 (3.50)	15.64 (0.06)	37.51 (negligible)
Scenario 4	10.43 (266.72)	12.99 (57.02)	15.84 (3.24)	19.18 (0.05)	39.08 (negligible)

Appendix B: Trackage configuration of rail network from Indio, CA to Colton, CA