22nd International Congress on Modelling and Simulation, Hobart, Tasmania, Australia, 3 to 8 December 2017
mssanz.org.au/modsim2017

Iterative train scheduling in networks with tree topologies: a case study for the Hunter Valley Coal Chain

A. Mendes [a], **M. Jackson** [b], **M. Rocha de Paula** [b] **and O. Rojas** [b]

[a] *School of Electrical Engineering and Computing, The University of Newcastle, NSW, Australia*
[b] *Hunter Valley Coal Chain Coordinator, Broadmeadow, NSW, Australia*
Email: Alexandre.Mendes@newcastle.edu.au

Abstract: This work describes an optimisation method based on genetic algorithms to generate train schedules for the rail network under the coordinating responsibility of the Hunter Valley Coal Chain Coordinator in NSW. The network connects 3 coal export terminals to 31 load points and haulage distances can extend up to 364 km. The scheduling problem consists of finding a high-quality schedule for trains travelling from a terminal to a load point and back, respecting all constraints imposed by the network itself and the operational environment. Those constraints refer to a mix of single and double tracks, limited parking facilities along the tracks, loading capacity at the load points, as well as minimum spacing (headway) between trains.

The decision variables include the travel speeds at each section of the network and the amount of dwell time for each train at each parking facility along the route. To test our approach, a simplified model of the HVCCC network, with 3 terminals, 11 load points and 40 sections was used. The objective function is the minimization of the total travel times. A lower bound for that objective function was calculated with the trains travelling at maximum speed, and no constraints being applied.

Three scenarios were tested, with 15, 30 and 60 trains; and with different configurations of the genetic algorithm. The results are presented in the form of a table with a number of statistics related to the solutions found, namely average travel time (with standard deviation), plus shortest and longest travel times, and CPU times. Relative to the lower bounds, the gaps for the average trip time range between 14% and 50%, depending of the problem size. These initial results are encouraging, considering the complexity of the system, the number and complexity of constraints, and the CPU time required by the method. Finally, in the discussion section we indicate possible paths of future research.

Keywords: *Train scheduling, supply chain, coal transportation, optimisation, genetic algorithms*

1 INTRODUCTION

The Newcastle Port is located in New South Wales, Australia, and is the largest coal export facility in the world by volume. In 2016, there were over 22,000 train trips, from 35 mines in the Hunter region into three shipping terminals to serve over 1,400 vessels, for a total exported volume of over 160 million tons of coal. The Hunter Valley Coal Chain Coordinator[1] (HVCCC) is responsible for coordination this effort, from rail transportation to stockpiling and ship loading. With the overall goal of throughput maximization, the coordination involves 11 producers in the Hunter region, two track owners/operators and two terminal operators, and aims at producing a medium term production plan approved by all stakeholders.

One of the critical sub-problems associated to the Hunter coal supply chain is the coordination of train travels. Each train movement is associated to one or more orders, i.e. one or more ships scheduled to berth at given times in the near future. Those orders refer to specific types and quantities of coal that must be sourced from the nearby mines. A typical trip will start with an empty train leaving one of the terminals, refueling at a service station, heading to a load point, loading, and heading back to a terminal, into one of its dump stations. Decisions about the type and size of the train, departure terminal, destination load point and destination terminal are all made beforehand, based on the orders to be fulfilled and available resources. That information is the main input to the scheduling problem. Currently, the task of scheduling train trips is done manually by a team of engineers. This process is time consuming and inefficient, and leads to the under-utilisation of resources.

Train scheduling is a well-known problem in the area of logistics and there are many variants. The types of problems are closely related to the type of network and the task at hand. In *intra-city networks*, like those found in urban centres, there are two problems to be solved: routing and scheduling. Routing problems arise in highly connected networks, where there are several possible routes between any two locations (Gendreau et al. [2015]). In *inter-city networks*, however, sometimes there is only one route between locations – or at least one route is clearly preferable over the other possibilities – thus eliminating the need for routing. This is the case of the network addressed in this study. From the terminals to any of the load points in the Hunter region there is only one path to be followed and thus the only relevant problem is the scheduling of the individual train travels. This problem has been visited recently in the context of high-speed trains in China (Yang et al. [2016]), and the authors used a mixed integer linear programming model. The network was relatively small, with 20 sections, and all sections were double track. The method achieved optimal results for problems with up to 48 trains in less than 10 hours.

In 2014, Upadhyay and Bolia [2014] addressed a similar problem in the context of dedicated freight railway corridors in India. The authors proposed a mathematical model and a Simulated Annealing (SA)-based heuristic. The model failed to solve any of the real-life sized problems (i.e. with more than 40 sections) and the SA found good solutions in around two minutes. The complete network was a bit larger, with 60 sections, and all sections were double track, as well.

Another relevant work to our study has used genetic algorithms and artificial neural networks to solve the scheduling problem for passenger trains in single tracks (Dundar and Sahin [2013]). The network was also small, with 18 sections, and for problems with up to 17 trains (8 travelling upstream and 9 travelling downstream), the method took 286 seconds to converge. The method is relevant as it takes into account single tracks and the conflicts that arise when trains meet-pass each other. A second study that addresses single-track networks, deadlocks and meet-pass constraints is Li et al. [2014]. In this work, the authors consider a small network with 9 single track sections and 8 meet-pass facilities. The mathematical model finds feasible solutions within the time limit of 5 hours, with optimality gaps in the order of 10%-40%.

Compared to the studies described above, the scenario derived from the HVCCC operations is considerably more complex, with additional constraints for loading capacity and also topologies that are particular to this network (e.g. balloon loops containing one or more load points). Given the additional complexity and also that the network has a size comparable to the study by Upadhyay and Bolia [2014], the use of a metaheuristic is arguably the most appropriate approach. For a recent overview of train routing and scheduling problems and their variations, we refer the reader to the recent book Wang et al. [2016]. Next we present a detailed description of the HVCCC network and problem at hand.

[1] http://www.hvccc.com.au/

2 PROBLEM DESCRIPTION

2.1 HVCCC network

The HVCCC network connects 3 coal export terminals to 31 load points and extends for 380 km into the Hunter region. In is composed of a mix of single and double track sections, with double tracks extending from the terminals all the way until approximately 120 km inland. All other sections off the main line are single track, with passing loops available in regular intervals. Passing loops can accommodate different train lengths and they come in various configurations, with up to 3 parallel tracks. The load points are located close to the mines. Some of them are just off the main line, connected by a single track section, whereas others require long distances to be traversed once the train leaves the main line. Another important aspect is that all load points have a baloon loop configuration. That is, when train comes from the access track, it enters the loop in a predefined direction. At a certain point it slows down so the loading process can take place, and once it is finished, the train accelerates back into the same access track, now travelling in the opposite direction. Also, trains can wait both before and after the loading process took place, but no overtaking is allowed as load points do not have parking facilities. The real network also has a number of other infra-structure, such as refueling and maintenance stations; connecting tracks between the double tracks in regular intervals – so a train can overtake another on a double track without the use of passing loops; and balloon loops with 2 and 3 load points, plus parking facilities, i.e. the topology is very domain-specific. For more information about the network and the logistics of the coal transportation, we refer the reader to Frazer and Sutherland [2010] and ARTC [2016][2]. Most of these aspects are considered in our model, but some will be left for future research. Next, we describe the problem addressed in this study.

2.2 Model description

In this section we describe the input, decision variables, objective function and constraints of the train scheduling problem.

Input parameters. The problem receives the following information as input:

- *Arcs (or sections)*: Information about the length of each arc, single/double track status and adjacent arcs.
- *Nodes*: Information on whether the node represents a terminal, a load point or a passing loop. If the node represents a passing loop, it will also contain the number of parallel tracks and their lengths.
- *Jobs/Trains*: A list of jobs containing origin terminal node, load point node and destination terminal node, plus loading time. In addition, the model receives information about the train assigned to each job, i.e. length, maximum speed and ready time. By design, the number of jobs is the same as the number of trains.

Decision variables. Given the input parameters, each train is assigned a path, going from origin node to load point node, and finally to the destination node. This path will be used by the algorithm when determining the decision variables values. Notice that since the network has a tree structure, there is only one possible path for each train, and thus routing decisions are not part of the problem.

- *Speeds*: Speed of each train in each section of the respective paths. Section speeds can not be less than half of the maximum speed of the train.
- *Wait times*: Wait time in each passing loop. If a decision is made to wait at a given passing loop, that wait time can not be less than 10 minutes, to allow for engine re-start procedures to take place. There is no limit for maximum wait time.

Objective function. The objective is to minimize the total travel times. That is, the interval between the train's ready time and the time it concludes its trip, arriving at the destination terminal.

Constraints. The problem is subject to several constraints related to the operational procedures observed by HVCCC and network topology:

[2]http://www.artc.com.au/uploads/ARTC-2016-25-Hunter-Valley-Corridor-Capacity-Strategy-Final-Oct-2016.pdf

- *Meet-pass*: It is prohibited to have two trains travelling in opposite directions within the same *single track section*. In our model, a section is defined as a length of track without passing loops, thus, by construction, passing loops are located between adjacent sections only. For *double track sections*, there are no meet-pass constraints.
- *Overtaking*: Overtaking is only allowed if the train being overtaken is parked in a passing loop.
- *Headway*: Trains need to maintain a 5-minute headway at all times when travelling in the same direction.
- *Loading capacity*: Only one train can be loaded at a time in a load point.
- *Load point topology*: All load points have a balloon loop topology, and each balloon loop contains one load point. All load points have a passing loop in the access section, thus allowing waiting/overtaking.

2.3 Solution

A feasible solution to the problem specifies the speed of each train in each section of the network, and the wait times, in a way that no constraints are violated. In addition, a good quality solution will have a low objective function value, ideally with trains travelling at high speeds and stopping very few times. For illustrative purposes, in Figure 1 we show a typical solution for an example network with 2 terminals, 2 loadpoints and 7 nodes in total. All sections have single tracks. Ten trains have been scheduled in total and the diagram shows the dynamics of the system, with wait times, loading times and meet-pass situations occurring without collisions. The caption of Figure 1 describes how the diagram should be interpreted.

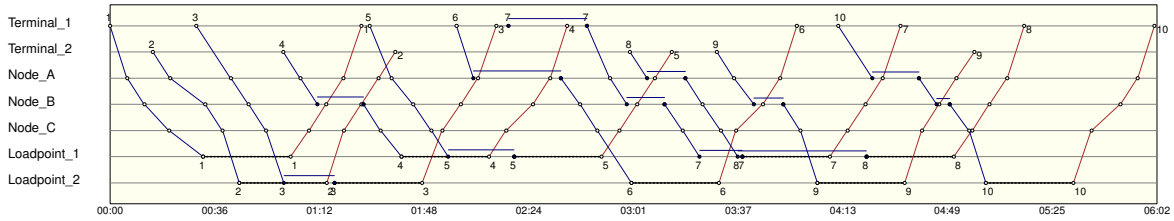

Figure 1. Diagram of a train schedule graph as generated by the method. The diagram show 2 terminals, 2 load points and 6 sections: ($Terminal\ 1 \leftrightarrow Node\ A$), ($Terminal\ 2 \leftrightarrow Node\ A$), ($Node\ A \leftrightarrow Node\ B$),($Node\ B \leftrightarrow Node\ C$), ($Node\ C \leftrightarrow Loadpoint\ 1$) and ($Node\ C \leftrightarrow Loadpoint\ 2$). All sections have single tracks and all nodes have passing loops. The thick, dashed lines at the terminals indicate loading operations, whereas the thin lines above and below the horizontal gridlines are wait times at passing loops. The correct way to read the diagram is as follows, using trains 1 and 4 as examples. Train 1 departs terminal 1 at 00:00, passes through nodes A, B and C without stopping, and reaches load point 1. It loads for 30 minutes and then goes back to terminal 1 without stopping. Train 4 departs from terminal 2, stops at $Node_B$ to allow trains 1 and 2 to meet-pass, continues to load point 1, loads and goes back to terminal 1. Trains 7 and 10 have wait times at terminal 1, which means they depart not at their ready times, but a few moments later to avoid collisions with the returning trains 4 and 7, respectively.

3 METHODOLOGY

In order to solve the train scheduling problem we use a genetic algorithm (Goldberg and Sastry [2010]) to search through the space of train speeds at each section, plus a greedy heuristic to eliminate unfeasibilities.

3.1 Genetic Algorithm

Genetic Algorithms (GA) are a population-based optimisation method. As with any metaheuristic, there is no guarantee that the optimal solution is found at all times, but a well-designed GA will achieve high-quality solutions in very short CPU times, making them a suitable alternate approach when exact methods are too time-consuming (Goldberg and Sastry [2010]). The characteristics of the GA implemented are as follows:

Representation. The GA uses a chromosome composed of integer values representing train speeds at each section. Each chromosome contains several arrays – one array per train – and the value in array i, position j, refers to the speed of train i at section j of its path.

Crossover and mutation. The crossover follows a Uniform Crossover (UX) strategy, i.e. for each position of the child's chromosome, the inherited value is chosen from the same position in one of the two parents. Then, the value copied goes through mutation, with a 5% probability of increasing by one and a 5% probability of decreasing by one. This is a relatively high level of mutation and aims at reducing premature convergence of the population.

Offspring acceptance. Any offspring with a better fitness value than one of its parents replaces the least fit parent. If the offspring is worse than both parents, it is discarded.

Population. Population size directly impacts diversity and computational complexity. Allowing more individuals to evolve will likely improve the quality of the result, but at the cost of larger CPU times. In our tests, we considered populations with 10, 20 and 50 individuals.

Fitness. The fitness of a solution is calculated using the speeds in its chromosome. Starting from the 1^{st} train, and all the way to the last one, the schedule is calculated with the trains travelling at their corresponding speeds for each section, immediately loading at the load point, and then returning to the destination terminals. It is clear that this procedure will lead to several unfeasibilities. Those unfeasibilities are solved by a greedy heuristic procedure. The fitness itself is the inverse of the objective function value, presented in Section 2.2.

3.2 Greedy heuristic

This heuristic aims at eliminating unfeasibilities related to the constraints listed under Section 2.2 and is described next:

Algorithm 1: Greedy algorithm to remove unfeasibilities

```
1  foreach (train) do //main loop
2      while checkLoadingConflict(train,terminal) == true do //checks if more than one train is loading at the terminal.
3          increaseWaitTime(train,terminal);
4      //loading time is now feasible for the train.  Proceed to check conflicts.
5      foreach (section ∈ train_path) do //checks each section of the path for unfeasibilities.
6          if (checkConflicts(train,section) == true) then //unfeasibility found
7              do
8                  waitTimeChanged = true;
9                  do
10                     increaseWaitTime(train,section);
11                     if (checkPassLoopConflict(train,section) == true) then //Passing loop is beyond its maximum capacity.
12                         removeWaitTime(train,section); //return to the original wait time value.
13                         section = previous(section); //go to the previous section.
14                         increaseWaitTime(train,section); //increase the wait time at the previous section.
15                     newUnfeasibilityCreated = false;
16                     foreach [(section' > section) ∧ (section' ∈ train_path)] do
17                         if [(checkConflicts(train,section') == true) ∨ (checkPassLoopConflict(train,section') == true) ∨
                           (checkLoadingConflict(train,terminal) == true)] then
18                             newUnfeasibilityCreated = true; //new unfeasibility was created at section'.
19                 while (newUnfeasibilityCreated == true) //no wait time change in the last checkout pass;
20             while (waitTimeChanged == true) //no wait time change in the last checkout pass;
```

The greedy algorithm works by checking and adjusting the schedule, one train at a time. For each train, it starts by checking if there is a loading unfeasibility, i.e. if the train is loading at the same time as another, at the same load point (line 2, Algorithm 1). If that happens, then wait times are added at the load point node until the unfeasibility is removed. Then, the procedure moves on to checking other conflicts, i.e. meet-pass in single tracks for trains travelling in opposite directions and headway distance for trains travelling in the same direction. That is achieved by checking each section in the train's path for conflicts with other trains (line 5). When a conflict is found (line 6), the wait time before entering the current section is increased until the conflict is removed. At this stage three problems might arise. First, having an extra train stopping at the passing loop might violate its capacity, and in this case the algorithm moves the wait time to one of the previous sections (lines 13 and 14). Second, while adding wait time might eliminate an existing conflict, other conflicts might be created in the later stages of the schedule. Finally, adding wait times in the first phase of the trip, i.e. from terminal to load point, might create loading unfeasibilities. That is the reason for the additional checking done in line 17. Once the schedule for the current train becomes feasible the method proceeds to the next train. That is, the algorithm is greedy from the sequence of trains perspective, with limited back-tracking occurring when wait times have to be added to a previous section due to a passing loop unfeasibility.

3.3 Iterative scheduling

The search space for the GA and the greedy heuristic grows quickly with the number of trains, thus impacting total CPU time. To address that issue, the GA iterates from the first to the last train, using a moving window, thus operating over a reduced search space in each iteration. For a window size of k, the GA will only schedule trains $[i - k, i + k]$ in each iteration, where i is the index of the current iteration. Trains outside that interval will either have their schedule fixed (if they have already been scheduled in a previous iteration), or have not been scheduled, yet (and thus do not influence the objective value at the current iteration). In the computational tests we compare the method's performance with different window sizes.

4 COMPUTATIONAL RESULTS

The algorithm was tested on a scaled-down version of the network described in Section 2.1, with 3 terminals, 11 load points and 40 sections. A lower bound is used for comparison purposes, calculated with the trains travelling at maximum speed and no constraints in the system. Next, we present a series of results for different sizes of problems (up to 60 trains), sizes of population for the GA (up to 50 individuals), and window sizes (up to 5). In all tests, passing loops along the network can only hold one train, whereas there is no maximum capacity for passing loops located at terminals and load points. All constraints from Section 2.2 are considered.

The results in Table 1 indicated that the GA is able to address problems with 60 trains in under 4 minutes (tests were run in an Intel i7 system). Relative to the lower bounds for the instances with 15, 30 and 60 trains, the gap for the average trip time is approximately 14%, 29% and 50%, respectively. Even though those seem to be high numbers, it is worth noting that the lower bound is based on a very naive procedure, and without any constraints imposed. Under that light, the results are indeed positive, considering the complexity of the system and the CPU time required by the method. As expected, larger populations have a positive impact on the results; and the best window size was 5 – the impact on CPU time was small, but the quality of solutions was considerably better compared to window sizes of 1 and 2.

4.1 Conclusions and Recommendations

This work addressed a train scheduling problem that arises in the context of the Hunter Valley Coal Chain Coordinator operations. The problem is more complex and considerably more constrained that others found in the literature. A genetic algorithm was implemented focusing on the train speed in each network section, with the unfeasibilities being eliminated through a greedy algorithm that adds wait times at passing loops for individual trains. The method runs quickly, with high quality solutions being found in less than 4 minutes. Results were compared against a lower bound, calculated with the trains travelling at maximum speed, and no constraints. The results are encouraging and additional constraints will be added to the method in the future. Those constraints will deal with limited (and absence of) parking facilities at terminals and load points, and refuelling operations. Another topic of further investigation is the iterative aspect of the method and the use of windows. A third line of research will be the development of a post-processing procedure to eliminate redundant wait times introduced by the greedy algorithm, aiming at shorter trips and a more realistic schedule that can be used by HVCCC's modelling team.

REFERENCES

ARTC (2016). 2016-2025 Hunter Valley corridor capacity strategy. Technical report, Australian Rail Track Corporation, Adelaide, Australia.

Dundar, S. and I. Sahin (2013). Train re-scheduling with genetic algorithms and artificial neural networks for single-track railways. *Transportation Research Part C 27*, 1–15.

Frazer, T. and S. Sutherland (2010). The Hunter Valley - a high capacity railway infrastructure network. In *Proceedings of CORE 2010: Rail, Rejuvenation and Renaissance. Wellington, New Zealand*, pp. 588–597.

Gendreau, M., G. Ghiani, and E. Guerriero (2015). Time-dependent routing problems: A review. *Computers and Operations Research 64*, 189–197.

Goldberg, D. and K. Sastry (2010). *Genetic Algorithms: The Design of Innovation* (2nd ed.). Springer, USA.

Li, F., J.-B. Sheu, and Z.-Y. Gao (2014). Deadlock analysis, prevention and train optimal travel mechanism in single-track railway system. *Transportation Research Part B: Methodological 68*, 385–414.

Table 1. Results of the genetic algorithm for different sizes of train scheduling problems. The table shows the results for the train travels, under different parameter configurations, namely the GA's population size and window size. (*) The sum of completion times includes wait times at the terminal of origin, i.e. the wait time between the train's ready time and the start of the trip. The other time measures (average trip time, shortest trip and longest trip) consider the time when the train leaves the terminal. That difference is important to allow for a direct comparison with the lower bound values.

Instance with 15 trains

Population size	Window size	Sum of completion times (hours) *	Average trip time (hours)	Standard Deviation	Shortest trip (hours)	Longest trip (hours)	CPU time (sec)
Lower bound		–	08:01	02:20	04:14	12:31	–
10	1	215.1	09:28	03:24	04:15	15:40	8.3
	2	212.7	09:23	03:14	04:15	15:04	9.7
	5	205.2	09:07	03:06	04:16	16:08	11.0
20	1	212.1	09:16	03:19	04:15	15:40	15.8
	2	210.4	09:14	03:13	04:15	14:54	14.7
	5	204.8	09:06	03:05	04:15	16:06	20.0
50	1	209.4	09:10	03:09	04:15	14:54	42.8
	2	210.6	09:15	03:15	04:15	14:56	47.3
	5	203.8	09:02	02:40	04:48	14:55	67.3

Instance with 30 trains

Population size	Window size	Sum of completion times (hours) *	Average trip time (hours)	Standard Deviation	Shortest trip (hours)	Longest trip (hours)	CPU time (sec)
Lower bound		–	08:26	02:15	04:14	13:33	–
10	1	710.5	11:51	03:59	04:15	18:59	40.5
	2	706.2	11:51	04:11	04:15	20:19	41.1
	5	662.2	11:21	03:45	04:16	19:09	45.2
20	1	666.7	10:58	03:17	04:15	16:29	101.2
	2	668.4	11:18	03:43	04:15	19:32	92.0
	5	638.5	10:51	03:21	04:22	17:18	105.5
50	1	661.0	11:01	03:27	04:15	17:24	517.6
	2	665.7	11:20	03:42	04:15	19:02	453.6
	5	617.2	10:52	03:31	04:15	20:32	463.2

Instance with 60 trains

Population size	Window size	Sum of completion times (hours) *	Average trip time (hours)	Standard Deviation	Shortest trip (hours)	Longest trip (hours)	CPU time (sec)
Lower bound		–	08:40	02:16	04:14	13:33	–
10	1	2,551.1	13:20	04:27	04:15	27:55	216.8
	2	2,482.9	13:08	04:26	04:15	28:18	218.2
	5	2,380.7	13:07	04:31	04:16	30:30	199.9
20	1	2,349.6	12:56	04:11	04:15	26:23	708.6
	2	2,350.9	12:37	03:56	04:15	23:37	640.3
	5	2,283.2	12:09	03:42	04:15	20:31	534.2
50	1	2,320.2	12:39	03:58	04:15	22:49	3,533.9
	2	2,287.6	12:16	03:23	04:15	23:14	3,208.4
	5	2,242.4	12:34	03:50	04:15	23:05	2,959.1

Upadhyay, A. and N. Bolia (2014). Combined empty and loaded train scheduling for dedicated freight railway corridors. *Computers and Industrial Engineering 76*, 23–31.

Wang, Y., B. Ning, T. van den Boom, and B. D. Schutter (2016). *Optimal Trajectory Planning and Train Scheduling for Urban Rail Transit Systems*. In: Advances in Industrial Control. New York: Springer.

Yang, L., J. Qi, S. Li, and Y. Gao (2016). Collaborative optimization for train scheduling and train stop planning on high-speed railways. *Omega 64*, 57–76.