An efficient genetic algorithm for the train scheduling problem with fleet management

Claudio Sanhueza
Hunter Valley Coal Chain Coordinator
Broadmeadow, NSW, Australia
claudio.sanhueza@hvccc.com.au

Alexandre Mendes
School of Electrical Engineering and Computing
The University of Newcastle
Callaghan, NSW, Australia
alexandre.mendes@newcastle.edu.au

Martin Jackson
Hunter Valley Coal Chain Coordinator
Broadmeadow, NSW, Australia
martin.jackson@hvccc.com.au

Riley Clement
Hunter Valley Coal Chain Coordinator
Broadmeadow, NSW, Australia
riley.clement@hvccc.com.au

***Abstract*—The Hunter Valley coal chain, located in New South Wales, Australia, is one of the most complex supply chains in the world. Coal orders are moved from the mines in the region to the terminals using a specific, complex rail infrastructure. These operations are scheduled by an experienced planning team at the Hunter Valley Coal Chain Coordinator. In this study, we propose an improved Genetic Algorithm to address the train scheduling problem. Our model considers several real-life operational constraints present in the coal supply chain and includes the selection of trains from an available fleet. Using a rail network with most of the real Hunter Valley railway infrastructure, we evaluate the strategy on test instances generated from actual train operations between 2017 and 2018. The objective of our strategy is to minimize total travel times. The algorithm was evaluated on instances with sizes between 60 and 180 jobs, and results show that the method can reach high-quality solutions – i.e. similar or better than those being currently used – in less than 2 minutes for the smaller instances, and 20 minutes for the larger ones.**

***Index Terms*—Train scheduling, genetic algorithm, optimisation, coal supply chain**

I. Introduction

The Hunter Valley coal supply chain is one of the largest in the world by volume exported. It comprises 3 terminals, 13 producers, 35 mines, 450 km of tracks and two train fleet operators. Over 1800 vessels transit each year through the Port of Newcastle, Australia, transporting approximately 180 million tons of coal. The Hunter Valley Coal Chain Coordinator (HVCCC)[1] is responsible for synchronizing all coal chain operations, with a focus on maximizing transported coal volumes, and minimizing overall costs.

To maximize throughput and keep the continuous operation of the coal chain, train trips must be scheduled efficiently. A train trip consists of an origin terminal, a destination load point (generally located at a mine site) and a destination terminal. Each vessel that arrives at a terminal requires one or more specific types of coal, which must be sourced from the local mines. When a loaded train arrives at the terminal, the coal is placed on a stockyard and waits until the corresponding vessel arrives so that the loading process can take place.

Currently, the schedule of train trips is performed manually by a team of planners. The process is time-consuming and requires considerable expert knowledge, making it very difficult for less-experienced planners, and virtually non-reproducible. Developing an efficient optimization algorithm to address that task is a natural step towards a better operational environment.

In this study, we propose an extended iterative train scheduling optimizer to compute feasible train trips in the context of the Hunter Valley coal chain. Specifically, our strategy extends the genetic algorithm proposed in [1] and presents the following additional contributions:

1) A more efficient genetic algorithm to compute feasible train schedules.
2) Train schedules are calculated without considering a reference timetable, allowing higher flexibility.
3) Incorporation of train fleet constraints in the model.
4) Testing using HVCCC's entire rail network and several real-life instances extracted from historical train trips.

We organise the remainder of the paper as follows. In Section II, we present a literature review, with an emphasis on train scheduling and routing. Next, in Section III, we discuss the Hunter Valley coal chain and present the problem statement. We describe the algorithm in Section IV, followed by the discussion about the experimental methodology and computational results in Section V. We finish with the conclusions and future work in Section VI.

II. Related Work

Scheduling is one of the most common tasks in any supply chain, and refers to the time allocation of resources to meet specific demands. Scheduling focuses on achieving specific objectives while maintaining safe operations and observing business constraints. The Train Scheduling Problem (TSCP)

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

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

is a complex optimization problem that has been proven NP-hard [2] [3]. Finding optimal solutions is rarely achievable in large-sized instances. There are many variants of the problem, based on specific characteristics, requirements, and problem-specific constraints. The solution approach is highly dependent on the features of the specific rail network. For example, highly connected rail networks lead to both routing and scheduling problems, since there are several possible paths between any two nodes. On the other hand, in networks with tree topologies, there is only one route between any two locations, so the routing problem is eliminated. That is the case of the Hunter Valley rail network, which has a single routing option between each load point and a terminal.

The TSCP has been addressed under many different scenarios, generating extensive literature. One of the first studies was presented in [4], where the a single-track TSCP was modeled as a mixed integer program. The author proposed a branch-and-bound method to compute a timetable minimizing total transit time. In the early 90's, Jovanovic and Harker [5] presented a nonlinear integer programming model, where the goal was to minimize the deviation between actual schedule and planned schedules. A branch-and-bound procedure was implemented to generate feasible meet-pass train plans. The problem variation that considers single- and double-tracks was addressed by Carey [6] [7] via a decomposition method. The author presented several node branching, variable fixing, and bounding approaches to reduce the search space. Later, the same author dealt with the scheduling and overtaking problems of trains with different speeds on the single-track rail line in [8].

More recently, in the 2000's, Zhou and Zhong [9] addressed a double-track train scheduling problem with multiple objectives. They implemented a branch-and-bound algorithm with an effective dominance rule and a beam search procedure with utility evaluation rules. The approach was evaluated using a Beijing-Shanghai high-speed railroad case study. Caprara et al. [10] developed a heavily-constrained mathematical model for a fundamental train timetabling problem using a Lagrangian heuristic. Later, Liu and Kozan [11] proposed a Blocking Parallel-machine Job-shop Scheduling (BPMJSS) based approach for train scheduling problems. In the model, trains correspond to jobs, single-track sections are equivalent to single machines, multiple-track sections represent parallel machines, and an operation is regarded as the movement/traversal of a train across a section.

All the previous studies used mathematical models for the problem, and either exact methods, or heuristics that allow an optimality gap to be calculated. These approaches limit the size of the instances that can be solved in short CPU times, and sometimes the number and/or complexity of the constraints included in the model. The next studies presented use metaheuristics-based approaches – either entirely, or as one component of the methodology. This allows them to address more complex, larger problems, but at the cost of losing the optimality guarantee (or the optimility gap).

Firstly, Dundar et al. [12] considered conflicts on a single-track infrastructure and proposed a genetic algorithm to address train scheduling with meet-pass conflicts. To evaluate the algorithm, artificial neural networks were implemented as a proxy to simulate the decision behaviour of train dispatchers, reproducing their conflict resolutions strategies. Sama et al. [13] dealt with the real-time problem of scheduling and routing trains in a French railway network. The train routing problem was formulated as an integer linear programming and solved via an ant colony algorithm to generate a subset of routes. The railway traffic problems receive as input the best subset of routing alternatives and is solved using a mixed-integer linear program. Recently, Xiao et al. [14] proposed a heuristic approach based on a genetic algorithm and tabu search to address a path-formulation of the block-to-train problem. The model was evaluated on small instances and applied to a real larger railroad sub-network in China. For a recent overview of train scheduling problems and studies, we refer the reader to the book published by Wang et al. [15]. Next, we present the problem addressed in this work.

III. Problem statement

Vessels that berth at the Port of Newcastle require specific coal types and quantities to satisfy their particular orders. Moving coal from mine to terminal requires planning, with train schedules playing a critical part. Train schedules are created daily by the planning team, allocating a suitable combination of resources to complete the orders.

Figure 1 shows a simplified representation of the main components of the coal chain. For a more detailed map of the railway infrastructure, we refer the reader to the websites of Port Waratah Coal Services[2] and the HVCCC[3]. In the next subsections, we describe relevant components of the supply chain, definitions, constraints and objectives.

A. Railway infrastructure

The Hunter Valley rail network has a very particular tree structure, which includes both double- and single-track sections. The mainline has dedicated up and down lines (i.e. double track sections), starting at the terminals and extending over 120 km inland. Sections outside the mainline are typically single-track, and are used for both up and down traffic. This configuration requires the use of passing loops, located at regular intervals, to allow trains moving in opposite directions to pass each other. Also, some areas of the rail network must be shared with other traffic such as passenger and freight (other than coal) trains. Along the mainline, however, typically there are tracks in both directions dedicated to coal traffic only. Next, we outline the key components of the rail network relevant to this work:

Passing loops – Passing loops are sections of track that enable trains travelling on the same line to overtake or meet-pass each other. Loops allow a single-line track to carry trains moving in opposite direction continuously. The coordination of these types of events requires a significant amount of planning

[2]https://www.pwcs.com.au/what-we-do/the-coal-chain/

[3]https://www.hvccc.com.au/AboutUs/Pages/MapOfOperationsV3.aspx

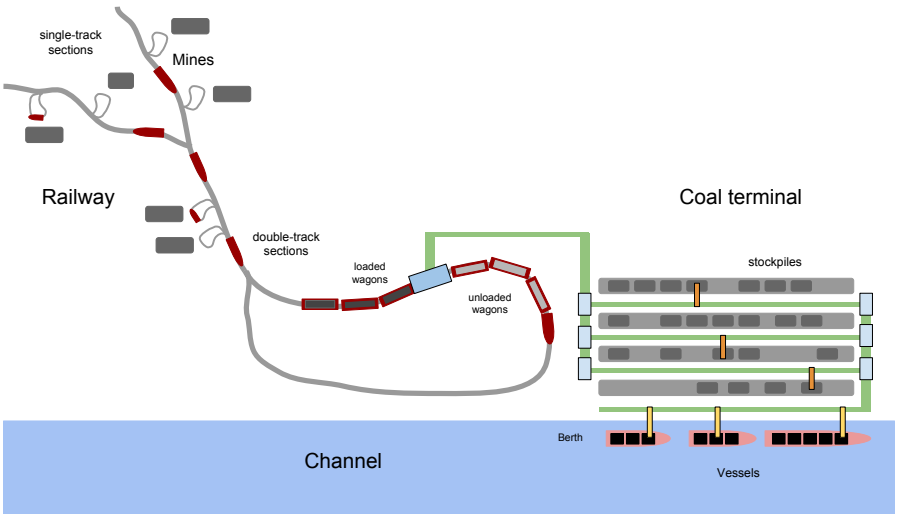

Fig. 1. The figure shows a diagram of the Hunter Valley coal chain showing a small subset of its components. Based on the demand, trains are scheduled to travel through the rail network moving coal from load points to the terminals. Trains that have arrived at the terminal are unloaded, and coal is moved onto specific stockpiles. Finally, vessels that are berthed are loaded with coal collected from corresponding stockpiles.

but guarantee that trains can keep moving without incidents or delays.

Load points – Load points are located near (or inside) mines and are responsible for loading coal onto the trains. While some load points are located just off the mainline, connected through a single track section, others are found far from the main line, requiring lengthy sections of tracks to be traversed. Load points are typically located inside balloon loops, which are sections of track where the two ends meet at a common point, allowing trains to change direction and return to the mainline. Trains are allowed to have wait times at the load point before the loading process starts, and after it is concluded. However, they cannot overtake other trains since load points do not have parking facilities.

Dump stations – Trains carrying coal usually have either side- or bottom-dumping wagons. The terminals' dump stations in the Hunter Valley coal chain are configured for bottom dumping. When trains reach a dump station, they slow down, the doors on the base of the wagon open, and the coal falls into steel containers, named hoppers. The coal is then fed onto conveyors that move it towards the stockyards.

Additional facilities – Some rail loops have additional facilities where train crews can change, and the locomotives can receive provisions such as fuel, sand, and water. Also, loops may have extra facilities for washing locomotives, and for performing regular inspections and maintenance services.

B. Train scheduling model

In this subsection, we present the main elements of our model, including inputs parameters, decisions variables, objective function and constraints.

Input parameters – Each instance of the train scheduling problem in the context of the HVCCC receives the following data as input:

1) Rail network: we represent the rail network as a graph $G(V, E)$, where V is the set of nodes and E corresponds to the edges. There are three types of nodes: terminal, load point, and passing loop. If a node represents a passing loop, it will contain information about the number of parallel tracks and their lengths. For nodes representing load points, the model considers the *recharge time*, which indicates the time required to reset the equipment, during which no trains can be loaded. Edges represent sections connecting two nodes in the network and are defined by a length, type (single- or double-track), the number of tracks (up and down), and maximum traversal speeds (up and down).
2) Jobs: a list of the jobs that need to be completed by a train fleet. They contain origin terminal node, load point node, destination terminal node, and the amount of coal that must be transported (in tons). Each job also has a ready time (i.e. the time when a job can be assigned to a train).
3) Train fleet: defines the set of train models that can be used to complete the jobs. Each train model is characterized by maximum speed when empty, maximum speed when loaded, length, amount of coal that it can transport (i.e. payload), and the number of trains of that model.

Decision variables – Given the input parameters, for each job the method chooses a train from the train fleet and computes the path for its trip. A path begins at a starting terminal, goes to a load point and returns to a destination terminal. 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 definition.

1) Trains: a suitable train for each job is selected from the available fleet. A train can be reused once it finishes the previous job, but cannot begin the next trip for at least 60 minutes, to account for provisioning and maintenance.

2) Speeds: the speed of each train in each section of its path depends on several factors. For instance, maximum speeds vary when trains are empty or loaded, to take into account physical and safety restrictions imposed by the rail authorities. In addition, when traversing a section, the speed of the train cannot be less than half of the maximum speed allowed.
3) Wait times: this refers to the wait time for each train in each passing loop. If a decision to stop at a given passing loop is made, that wait must be greater than 10 minutes, to allow for engine restart operations and brake system recharge. There is no limit for the maximum wait time.

Objective function – The objective is to minimize the summation of travel times, which is computed as the interval between the job ready time and the time the corresponding train concludes its trip, arriving at the destination terminal.

Constraints – The scheduling problem is subject to several constraints that reflect the operational environment of the HVCCC.

1) Meet-pass: two trains cannot travel in opposite directions within the same single-track section. In this model, sections represent tracks that do not have passing loops, and passing loops are positioned between adjacent sections. Double-track sections do not have meet-pass restrictions.
2) Overtaking: a train can overtake another one if and only if the latter is parked in a passing loop.
3) Headway: trains travelling in the same direction must maintain a 5-minute headway at all times.
4) Load points: All load points have a balloon loop topology, and each balloon contains one load point. Only one train can be loaded at a time in a load point. Load points have a recharge time between consecutive loading operations.
5) Contractual constraints: due to contractual agreements between members of the coal chain, there are limitations on the train models that can travel from particular load points to specific destination terminals. To model that, a list of constraints is used, with each constraint containing three elements: train model, load point, and destination terminal. Those elements indicate a combination that is not allowed under current agreements, thus limiting the candidate trains that can complete specific jobs.

Train scheduling solution – In this study, a solution corresponds to a set of train schedules (one per job). Each train schedule specifies the speed of the train in each section of the railway network and the wait times during the trip. Trains are chosen from a fleet of available train models. High-quality solutions will exhibit short travel times, fewer and shorter wait times and, of course, all constraints will be respected.

IV. Genetic Algorithm

The proposed algorithm is a greedy, iterative Genetic Algorithm (GA). It is an extended version of the algorithm presented in [1] that includes train fleet management, thus allowing the selection of specific train models to complete jobs. Genetic algorithms are population-based optimization strategies. While these methods are not guaranteed to find an optimal solution, a well-designed GA can compute high-quality solutions within short CPU times. Next, we present the main characteristics of the GA implemented in this study.

Representation – A solution is represented as an array of integers, which are mapped to the train's speed in each path section. The value $s_{i,j}$ corresponds to the speed of the train selected for job i at section j of its path. For example, if the minimum and maximum speeds are 40 and 75 km/h, the integer values would be $\{0, 1, ..., 7\}$ = $\{$40 km/h, 45 km/h, ..., 75 km/h$\}$, as train speeds are defined in 5 km/h intervals in our tests.

Population – The population of individuals is structured as a hierarchical, ternary tree, with recombination happening between parent and child nodes, only. This structure has been used in several works since 2001 [16] and is very efficient as it induces a parallel evolution behaviour in the branches of the three. This structure has performed better than unstructured populations (with the same number of individuals) in other problems such as the Travelling Salesman Problem [17]. For more information about the effect of population structures in evolutionary algorithms, we refer the reader to a recent survey of the area [18]. In our tests, the population size was 13 individuals, corresponding to a ternary tree with three levels.

Recombination and mutation – Recombination between two solutions uses the Uniform Crossover strategy. In this procedure, the value in each position of the offspring solution is inherited from the same position in one of the parents, at random. After crossover, we apply mutation to minimize premature convergence of the population. To do so, each value in the offspring goes through mutation with a 5% probability of increasing train speed by one step and a 5% probability of decreasing train speed by one step (i.e. 5 km/h).

Offspring acceptance – Any offspring with a better fitness value than one of its parents replaces the least fit parent. The method rejects an offspring solution if its fitness value is worse (or the same) than both parents. This strategy puts significant evolutionary pressure in the process, but that is counteracted by the strong mutation policy.

Fitness – The fitness of a solution is calculated using the speeds encoded in its chromosome. Starting from the first job, 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. This procedure will likely produce an infeasible schedule, and we solve those with a greedy feasibilization method, which we will describe soon. The fitness itself is the inverse of the objective function value, presented in Section III-B.

The pseudocode of the GA is presented in Algorithm 1. The algorithm is greedy in the sense that it works by scheduling a subset of the jobs in each iteration, using a sliding window approach. The procedure begins by building an initial population of individuals (Algorithm 1, line 3) using one of the two available speed profiles. The speed profile

parameter allows one to select either Only Train Speeds (OTS), or Train & Section Speeds (TSS). The OTS profile uses only the maximum speeds defined in the train fleet (i.e. unloaded and loaded maximum speeds) to build a schedule, ignoring the section speeds. The TSS profile, on the other hand, combines train speeds and section traversal speeds, choosing the maximum speed that a train can travel based on both constraints.

The method then creates the tree-structure population to store the solutions (Algorithm 1, line 4). A feasible solution is computed using the procedure COMPUTESCHEDULE (see Algorithm 2). It generates feasible train schedules for the jobs in the set J_w, which represents the jobs being currently scheduled for a window size w. The procedure receives a list of jobs to be completed and computes a set of candidate train models that can perform those trips. The train model chosen is that with the most trains available at a given ready time. If no train is available, the procedure selects the earliest possible train based on previous schedules. Next, the method computes the train's path including a detour to perform refueling operations if that is a requirement for the selected train (Algorithm 2, lines 5–7).

As mentioned before, simply using a solution's speeds on each path section will likely produce infeasible schedules. Our algorithm removes infeasibilities by checking each section in the train's path for conflicts with other trains (Algorithm 2, lines 8–21), and eliminates these infeasibilities using the procedure REMOVEINFEASIBILITIES (Algorithm 3).

REMOVEINFEASIBILITIES starts by checking loading infeasibilities, i.e. if the train is loading at the same time as another at the same load point, or if the recharge time is being violated. To remove loading infeasibilities, the algorithm iteratively adds wait times at the load point node (Algorithm 3, lines 1–3).

Then, the procedure checks for other conflicts, i.e. meet-pass in single tracks for trains travelling in opposite directions, and headway distance for trains going in the same direction. Every time the approach finds a conflict (Algorithm 3, line 4), the method increases the wait time before entering the current section until the conflict is removed.

At this point, three problems might arise. First, having an additional train stopping at a given passing loop might violate its capacity, and in this case, the algorithm moves the wait time to one of the previous passing loops (Algorithm 3, lines 9–10). Second, while adding wait times might eliminate an existing conflict, the strategy might create other conflicts in the later stages of the schedule. Finally, another issue is that adding wait times in the first phase of the trip, i.e. from a terminal to load point, might create load point infeasibilities. That triggers the flagging of an additional check for conflicts (Algorithm 3, line 17).

Once the schedule for the current train becomes feasible, the method proceeds to the next job, until all jobs in the current window have been scheduled. Then, the window is moved to the next set of unscheduled jobs and the GA starts again (but now with all previously scheduled jobs fixed).

Algorithm 1 Pseudo-code of the genetic algorithm for the train scheduling problem with train fleet management.

Input: rail network G, set of jobs J, train fleet T, population size p_s, job window size w, probability of crossover Pr_c, probability of mutation Pr_m, speed profile s_p
Output: train schedule S_J
1: $\mathcal{S}_J \leftarrow \emptyset$
2: $J_w \leftarrow$ GETJOBWINDOW(J, w) ▷ List of jobs to be scheduled in the next iteration
3: $P \leftarrow$ INITPOPULATION(G, J, T, p_s, s_p) ▷ Initializes population with a speed profile
4: $\mathcal{T}_P \leftarrow$ UPDATEPOPULATIONTREE(P) ▷ Hierarchical structure
5: $J_c \leftarrow \emptyset$ ▷ List of jobs completed is empty
6: **while** $J_w \neq \emptyset$ **do**
7: $P_c \leftarrow$ CROSSOVER$(\mathcal{T}_P, J_w, Pr_c)$
8: $P^* \leftarrow$ MUTATION(P_c, J_w, Pr_m)
9: COMPUTESCHEDULE$(S_i), \forall S_i \in P^*$ ▷ Calculates the fitness of the offspring solutions
10: **if** $P \neq P^*$ **then** ▷ If the population has changed
11: $\mathcal{T}_P \leftarrow$ UPDATEPOPULATIONTREE(P^*)
12: **else**
13: $P \leftarrow$ RESTARTPOPULATION(G, J, P^*) ▷ Resets the population
14: $\mathcal{T}_P \leftarrow$ UPDATETERNARYTREE(P)
15: **end if**
16: $\mathcal{S}_J \leftarrow \mathcal{S}_J \cup$ GETSCHEDULES$(\mathcal{T}_P, J_w)$ ▷ Updates the list of schedules
17: $J_c \leftarrow J_c \cup J_w$ ▷ Updates the list of jobs completed
18: $J_w \leftarrow$ GETJOBWINDOW$(J - J_c, w)$ ▷ Gets additional jobs to schedule
19: **end while**
20: **return** S_J ▷ Returns the train schedules

Algorithm 2 Pseudo-code to COMPUTESCHEDULE

Input: train fleet T, individual I, set of jobs J_w, speed step s
Output: a set of schedules S_{J_w}
1: $S_{J_w} \leftarrow \emptyset$ ▷ Starts with no schedules calculated
2: **for each** job $j \in J_w$ **do**
3: $time \leftarrow$ GETREADYTIME(j)
4: $S_j \leftarrow \emptyset$ ▷ Schedule of job j is reset
5: $T_j \leftarrow$ SELECTAVAILABLETRAIN(j, T)
6: UPDATEINITIALSPEEDS(I, T_j) ▷ Based on train speed profile
7: $P_j \leftarrow$ GETPATH(j, T_j) ▷ Includes refueling detour if T_j needs it
8: **for each** section $e \in P_j$ **do**
9: ADDENTRYTIMEEDGE$(S_j, e, time)$ ▷ Calculates the entry time in section e
10: $ht \leftarrow$ GETDISTANCE$(e)/($SPEED$(j, e) * s)$ ▷ Time for the head of the train to traverse section e
11: $tt \leftarrow ht +$ GETLENGTH$(T_j)/($SPEED$(j, e) * s)$ ▷ Time for the tail to traverse section e
12: $time \leftarrow time + ht$ ▷ Updates the current time of the job
13: ADDEXITTIMEEDGE$(S_j, e, time)$
14: $n_{dest} \leftarrow$ GETDESTINATIONNODE(e)
15: **if** ISLOADPOINT(n_{dest}) **then** ▷ Adds the loading time
16: $time \leftarrow time +$ GETLOADTIME(n_{dest})
17: **end if**
18: SETMAKESPAN$(S_j, time)$ ▷ Sets the makespan of job j
19: REMOVEINFEASIBILITIES(j, e, T_j) ▷ Removes infeasibilities from the schedule (Algorithm 3)
20: $S_{J_w} \leftarrow S_{J_w} \cup S_j$ ▷ Updates the list of jobs already scheduled
21: **end for**
22: **end for**
23: **return** S_{J_w}

V. EXPERIMENTS

Several computational experiments were conducted to evaluate the scheduling algorithm. The method itself was implemented in Java, and was tested on individual machines in the

Algorithm 3 Pseudo-code to REMOVEINFEASIBILITIES

Input: a job j, a section e, a train T_j
Output: feasible train schedule

while CHECKLOADINGCONFLICTS(j, T_j) **do** ▷ Load point capacity violation
  INCREASEWAITTIME(j, e, T_j) ▷ Increases wait time at section e
end while
if CHECKCONFLICTS(j, e) **then** ▷ Infeasibility found
  do
    INCREASEWAITTIME(j, e, T_j) ▷ Increases wait time at section e
    if CHECKPASSLOOPCONFLICTS(j, e, T_j) **then** ▷ Pass loop capacity violation
      REMOVEWAITTIME(j, e, T_j) ▷ Removes wait time at that pass loop
      $e \leftarrow$ GETPREVIOUS(e) ▷ Moves to a previous pass loop
      INCREASEWAITTIME(j, e, T_j) ▷ Increases wait time at the previous pass loop
    end if
    $infeasible \leftarrow$ false
    for each section $e^* \in$ GETPATH(j) **and** $e^* > e$ **do**
      if CHECKCONFLICTS(j, e^*) **or** ▷ Checks for new conflicts ahead along the path
        CHECKPASSLOOPCONFLICTS(j, e^*, T_j) **or**
        CHECKLOADINGCONFLICTS(j, T_j) **then**
        $infeasible \leftarrow$ true
      end if
    end for
  while $(infeasible)$ ▷ Repeat while schedule is infeasible
end if

University of Newcastle's Research Computing Grid, which consists of computers with Intel® Xeon® CPU E5-2698 v3 @ 2.30 GHz processors with 128 Gb of RAM in total.

A. Computational results

The network representation used in the tests is similar to the real Hunter Valley railway network infrastructure except for the sub-networks found inside the terminals. The network is composed of 3 terminals, 37 load points, two refuelling stations, 174 intermediate stations and 259 sections (i.e. a graph with 216 nodes and 259 edges). In addition, passing loops along the network can only hold one train, and there is no capacity limit for passing loops located at the terminals.

The job instances were created using real HVCCC train operations conducted between January 2017 and August 2018. From them, we extracted 9,000 complete cycles containing the required data to generate our experimental train scheduling instances. We evaluated the genetic algorithm with instances containing 60 to 180 jobs (i.e. 1 to 3 days of operations), in increments of 30 jobs. Based on the results obtained in the study presented by Mendes et al. [1], we set the population size to 50, and the jobs window size was set to 5.

The details of the train fleet is another input given to the method. We employ the same train fleet definition that is currently used on simulation tools developed by HVCCC's modelling team. The fleet is composed of 13 different train models. To evaluate our train fleet assignment procedure, we execute experiments with different fleet sizes, ranging from 25% to 100% of the real numbers (Table I).

In the experiments, after finishing a journey at a terminal, trains cannot begin a new journey before their recharge time is completed – that time was set to 1 hour. Also, refuelling operations are required whenever a train has traveled more than 100 km.

Table II shows the results of the genetic algorithm using the OTS profile and Table III shows the results using the TSS profile. Regarding the train fleet sizes, as expected, smaller fleets have a negative impact on the average makespan, in general. However, there appears to be an excess of trains in the system. Reducing the fleet to 50% affects the makespan only slightly (i.e. less than 10% increase). In that case, it would be worthy investigating whether the system can handle an increase in the current frequency of 60 train departures per day. It is likely that the train fleet can support such increase, but the network capacity might induce bottlenecks in other parts of the system.

The average, shortest and longest train trip values do not change substantially with the number of jobs and train fleet sizes. Therefore, one can conclude that the system has excess capacity running at a rate of 60 departures per day. Moreover, when there are not enough trains available, jobs just take longer to start – but when a train starts its trip, it faces low levels of congestion along its path. The average makespan increases, but proportionally to the number of jobs, which is the expected (and desired) result.

The GA can compute feasible train schedules in less than 2 minutes on instances with 60 jobs. For 180 jobs, the method took 15-20 minutes to find good solutions. It is important to notice that CPU times depend on the amount of overlap between train trips. That is, conflicts between train schedules need only to be checked for trains currently travelling in the network. Trains that have already completed their trips do not add to the complexity of the problem.

In addition to makespan and average trip times, one useful way to evaluate the results of the algorithm is to compare

TABLE I
THE TABLE SHOWS THE DETAILS OF THE TRAIN FLEET USED IN THE EXPERIMENTS. EACH ROW INDICATES A SPECIFIC TRAIN MODEL AND THE NUMBER OF TRAINS AVAILABLE PER MODEL. THE TRAINS ARE OWNED BY FOUR RAIL HAULAGE OPERATORS: PACIFIC NATIONAL (PNT), AURIZON (AZN), FREIGHTLINER AUSTRALIA (FLA), AND SOUTHERN SHORTHAUL RAILROAD (SSR). ON THE RIGHT-HAND SIDE, WE HAVE THE NUMBER OF TRAINS AVAILABLE FOR THE JOBS UNDER DIFFERENT FLEET SIZES – 25%, 50%, 75%, AND 100% OF THE REAL NUMBERS.

Train model	Length (meters)	Payload (tonnes)	Number of trains 100%	75%	50%	25%
FLA92	1,400	9,200	8	6	4	2
PNT91	1,380	9,100	8	6	4	2
PNT86	1,300	8,600	4	3	2	1
PDT86	1,300	8,600	3	2	1	0
AZN86	1,300	8,600	10	7	5	2
AZN80	1,200	8,000	5	3	2	1
PNT79	1,200	7,900	12	9	6	3
PNT68	1,000	6,800	2	1	1	0
PNT41	610	4,150	4	3	2	1
FLA42	630	4,150	2	1	1	0
SSR39	600	3,900	3	2	1	0
PNT33	500	3,300	2	1	1	0
PNT22	300	2,200	2	1	1	0

TABLE II
RESULTS OF THE GA USING THE OTS PROFILE. THE TABLE PRESENTS THE RESULTS OF THE GENETIC ALGORITHM FOR DIFFERENT SIZES OF INSTANCES AND TRAIN FLEETS. THE FIRST THREE COLUMNS OF RESULTS REPORT AVERAGE, FASTEST AND SLOWEST TRAVEL TIMES. THE TABLE ALSO REPORTS THE AVERAGE MAKESPAN FOR THE EACH COMBINATION OF JOBS/FLEET SIZE, AND THE CPU TIMES.

Jobs / fleet size (%)	Average trip (hh:mm)	Fastest trip (hh:mm)	Slowest trip (hh:mm)	Average makespan (hh:mm)	CPU time (sec)
60 / 100	14:12	5:06	22:08	115:33	76.2
60 / 75	14:12	5:05	21:59	111:25	72.1
60 / 50	14:10	5:04	21:55	117:06	69.4
60 / 25	13:59	5:03	21:07	142:31	56.4
90 / 100	14:21	4:47	22:51	167:05	201.2
90 / 75	14:19	4:47	22:35	167:50	196.1
90 / 50	13:54	4:47	21:40	175:46	152.7
90 / 25	14:02	4:47	21:43	211:19	150.3
120 / 100	14:24	4:37	23:24	228:22	456.5
120 / 75	14:15	4:36	23:14	234:05	384.1
120 / 50	14:11	4:36	23:04	238:08	368.6
120 / 25	13:56	4:37	22:04	284:03	322.3
150 / 100	14:26	4:29	23:51	286:03	759.0
150 / 75	14:12	4:29	23:36	295:06	690.8
150 / 50	14:18	4:30	23:20	288:38	673.2
150 / 25	13:56	4:29	22:27	347:48	598.3
180 / 100	14:20	4:32	23:43	341:36	1161.8
180 / 75	14:16	4:32	23:45	344:22	1158.3
180 / 50	14:19	4:30	23:43	371:35	1090.1
180 / 25	14:06	4:32	22:58	416:22	1082.4

travel times from the terminals to specific load points and back. In Figure 2, we show a comparison between the distribution of those times for each load point. There are three sources of data: the scheduled trips (the values estimated in the scheduling planning phase by HVCCC's planners), and the GA with the two speed profiles (OTS and TSS). The blue boxes refer to the scheduled trips, and the orange and green boxes refer to the GA with OTS and TSS profiles, respectively. As mentioned before, the scheduled data was collected from trips between January 2017 and August 2018, directly from HVCCC's database. Out of the 37 load points, only 19 are shown in Figure 2, as they represent those with complete data – many entries in the database had errors or missing values, and those 19 were the load points with the most trustful results over the 20-month interval.

The distribution of travel times generated by the two GA-based methods is very similar, and in most cases slightly better than the scheduled ones, which indicates that the model is reflecting the constraints and properties present in the real network. The planning team at HVCCC currently uses a proprietary, software-based integrated planning system to create a 24 hour (60 jobs) schedule. The process requires approximately 8 hours and a team of three people. The GA approach completes the same task in a fraction of the time, with similar or better results.

VI. CONCLUSIONS

This study addressed a train scheduling problem faced by the Hunter Valley Coal Chain Coordinator and proposed a genetic algorithm to compute feasible train schedules.

The scheduling problem is complex and more constrained compared to others found in the literature. The genetic algorithm generates solutions by assigning a train speed for each network section, with the infeasibilities being eliminated through a greedy procedure that adds wait times at passing loops. The method also selects trains for jobs from an available train fleet. The strategy is very efficient, with high-quality solutions being found in less than 2 minutes for smaller instances, and less than 20 minutes for larger ones. The method was tested on a large set of test instances generated using historical data from the HVCCC. The results are positive, with the distribution of travel times generated by the genetic algorithm being slightly better than the scheduled ones. That indicates that the model is reflecting the conditions present in the real network.

Additional constraints can be added to the method in the future. Among those, constraints to model rail maintenance operations are a prime candidate. Typically, maintenance activities are scheduled events, but they can also be triggered due to unexpected disturbances such as extreme weather conditions and accidents. Another path for future research is the inclusion of constraints on resources located at the terminals. Each terminal has a very particular infrastructure and a railway subnetwork that trains need to traverse, usually at low speeds, to perform unloading operations. The inclusion of such constraints will generate more realistic schedules that can be used by HVCCC's modelling team to make more accurate planning, strategic and operational decisions. Finally, another potential extension is to consider additional objectives (e.g. dwell times, operational costs and coal throughput). In this case, the problem might require a multi-objective optimization strategy, which will pose a series of new challenges.

TABLE III
RESULTS OF THE GA USING THE TSS PROFILE. SIMILARLY TO TABLE II, THIS TABLE PRESENTS THE RESULTS OF THE GENETIC ALGORITHM FOR DIFFERENT SIZES OF INSTANCES AND TRAIN FLEETS.

Jobs / fleet size (%)	Average trip (hh:mm)	Fastest trip (hh:mm)	Slowest trip (hh:mm)	Average makespan (hh:mm)	CPU time (sec)
60 / 100	13:43	4:52	22:00	107:17	68.5
60 / 75	13:47	4:49	21:58	108:15	67.1
60 / 50	13:41	4:46	21:37	110:39	64.1
60 / 25	13:30	4:48	20:54	139:23	54.6
90 / 100	13:51	4:37	22:21	162:03	173.8
90 / 75	13:52	4:36	22:34	163:42	167.2
90 / 50	13:51	4:33	22:17	166:18	160.1
90 / 25	13:36	4:35	21:35	202:54	141.3
120 / 100	13:58	4:26	23:20	217:33	362.4
120 / 75	13:54	4:24	23:08	220:37	366.5
120 / 50	13:53	4:23	23:10	214:51	347.7
120 / 25	13:36	4:24	22:07	267:37	308.6
150 / 100	13:58	4:15	23:39	275:25	669.6
150 / 75	13:48	4:14	23:23	271:41	612.2
150 / 50	13:53	4:14	23:13	276:55	626.2
150 / 25	13:37	4:23	22:29	334:53	577.3
180 / 100	12:44	3:59	22:28	324:41	841.3
180 / 75	13:24	4:08	23:14	319:40	940.5
180 / 50	13:33	4:16	23:14	336:29	991.5
180 / 25	13:30	4:20	22:36	405:44	989.5

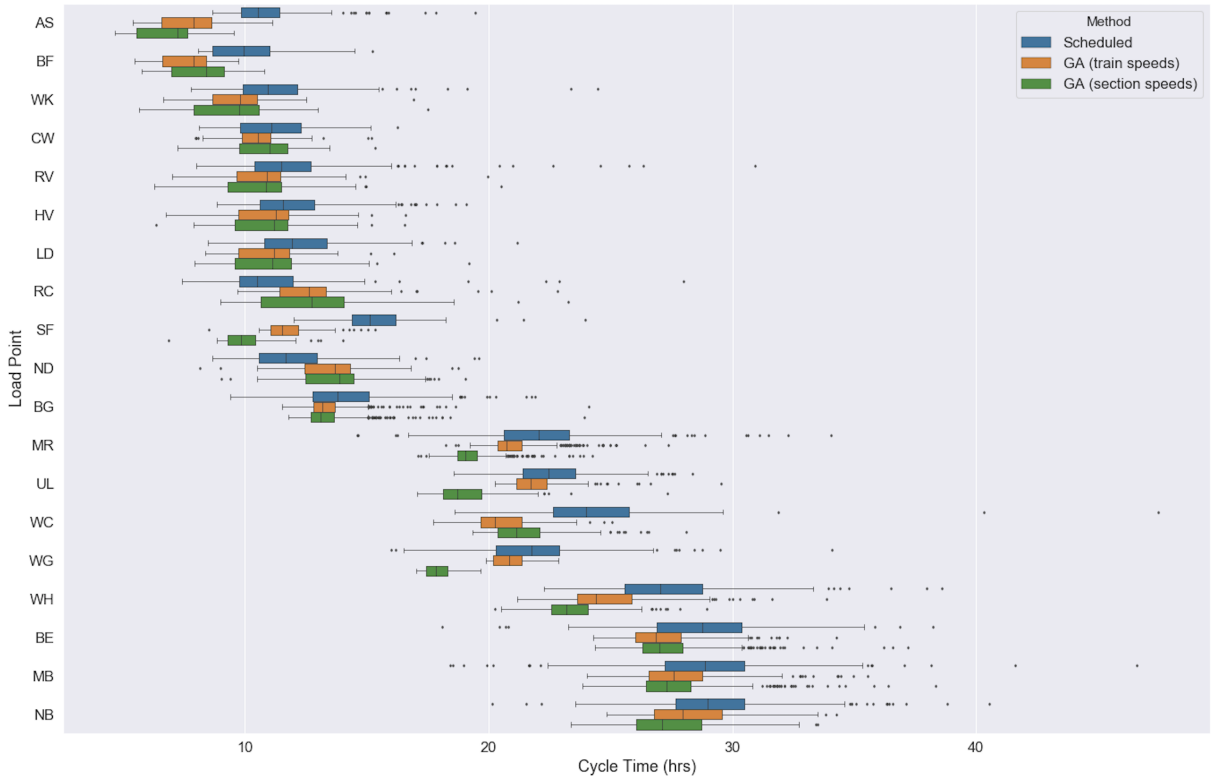

Fig. 2. This figure shows a comparison of trip times distribution per destination load point. Nineteen load points are shown, and are ordered from closest to farthest from the terminals. The blue box plots correspond to the scheduled data from HVCCC real-world operations, prepared by the company's planning team. The orange and green box plots correspond to the GA using the OTS and the TSS profiles, respectively. Note that the GA trips generally have a similar time distribution, or are slightly better than the scheduled ones.

ACKNOWLEDGMENT

This work was supported by the Innovation Connections grant number ICG000547, from Business.gov.au, Commonwealth of Australia.

REFERENCES

[1] A. Mendes, M. Jackson, M. R. de Paula, and O. Rojas, "Iterative train scheduling in networks with tree topologies: a case study for the hunter valley coal chain," in *Proceedings of the 22nd International Congress on Modelling and Simulation (MODSIM2017)*, 2017, pp. 1337–1343.

[2] X. Cai, C. Goh, and A. I. Mees, "Greedy heuristics for rapid scheduling of trains on a single track," *IIE transactions*, vol. 30, no. 5, pp. 481–493, 1998.

[3] A. Caprara, M. Fischetti, and P. Toth, "Modeling and solving the train timetabling problem," *Operations research*, vol. 50, no. 5, pp. 851–861, 2002.

[4] B. Szpigel, "Optimal train scheduling on a single line railway," *Operational Research*, vol. 34, no. 4, pp. 343–352, 1973.

[5] D. Jovanović and P. T. Harker, "Tactical scheduling of rail operations: the scan i system," *Transportation Science*, vol. 25, no. 1, pp. 46–64, 1991.

[6] M. Carey, "Extending a train pathing model from one-way to two-way track," *Transportation Research Part B: Methodological*, vol. 28, no. 5, pp. 395–400, 1994.

[7] ——, "A model and strategy for train pathing with choice of lines, platforms, and routes," *Transportation Research Part B: Methodological*, vol. 28, no. 5, pp. 333–353, 1994.

[8] M. Carey and D. Lockwood, "A model, algorithms and strategy for train pathing," *Journal of the Operational Research Society*, vol. 46, no. 8, pp. 988–1005, 1995.

[9] X. Zhou and M. Zhong, "Bicriteria train scheduling for high-speed passenger railroad planning applications," *European Journal of Operational Research*, vol. 167, no. 3, pp. 752–771, 2005.

[10] A. Caprara, M. Monaci, P. Toth, and P. L. Guida, "A lagrangian heuristic algorithm for a real-world train timetabling problem," *Discrete applied mathematics*, vol. 154, no. 5, pp. 738–753, 2006.

[11] S. Q. Liu and E. Kozan, "Scheduling trains as a blocking parallel-machine job shop scheduling problem," *Computers & Operations Research*, vol. 36, no. 10, pp. 2840–2852, 2009.

[12] S. Dündar and İ. Şahin, "Train re-scheduling with genetic algorithms and artificial neural networks for single-track railways," *Transportation Research Part C: Emerging Technologies*, vol. 27, pp. 1–15, 2013.

[13] M. Sama, P. Pellegrini, A. D'Ariano, J. Rodriguez, and D. Pacciarelli, "Ant colony optimization for the real-time train routing selection problem," *Transportation Research Part B: Methodological*, vol. 85, pp. 89–108, 2016.

[14] J. Xiao, J. Pachl, B. Lin, and J. Wang, "Solving the block-to-train assignment problem using the heuristic approach based on the genetic algorithm and tabu search," *Transportation Research Part B: Methodological*, vol. 108, pp. 148–171, 2018.

[15] Y. Wang, B. Ning, T. Van den Boom, and B. De Schutter, *Optimal trajectory planning and train scheduling for urban rail transit systems*. Springer, 2016.

[16] P. M. Franca, A. Mendes, and P. Moscato, "A memetic algorithm for the total tardiness single machine scheduling problem," *European Journal of Operational Research*, vol. 132, no. 1, pp. 224–242, 2001.

[17] L. Buriol, P. M. Franca, and P. Moscato, "A new memetic algorithm for the asymmetric traveling salesman problem," *Journal of Heuristics*, vol. 10, no. 5, p. 483–506, 2001.

[18] P. Moscato and C. Cotta, "An accelerated introduction to memetic algorithms," in *Handbook of Metaheuristics*. Springer, 2019, pp. 275–309.