Optimising a Real-time Scheduler for Railway Lines using Policy Search

Rohit Prasad, Shivaram Kalyanakrishnan
Indian Institute of Technology, Bombay
Mumbai, India
[rohitrp,shivaram]@cse.iitb.ac.in

Harshad Khadilkar
TCS Research
Mumbai, India
harshad.khadilkar@tcs.com

ABSTRACT

This paper describes a policy search approach for railway scheduling using the covariance matrix adaptation evolution strategy (CMA-ES). The goal is to define arrival/departure times and track allocations for all trains such that the resource and operating constraints of the railway line are satisfied, and priority-weighted train departure delay is minimised. The proposed approach is scalable in the sense that (i) the optimised policy can be applied to an arbitrarily long railway line, independent of the number of trains, tracks, and stations, and (ii) the on-line implementation is computationally light enough to be applied in real-time. Our experiments show that policies computed with CMA-ES are able to produce solutions with lower priority-weighted delay than heuristics and reinforcement learning (RL) algorithms reported in literature, on synthetic examples as well as actual railway line data from portions of the Indian Railway network.

KEYWORDS

Evolutionary Policy Search; Planning; Railway Scheduling

1 INTRODUCTION

Punctuality is one of the principal factors for users (both passenger and freight) when selecting a mode of transport, in addition to cost and time considerations. Road and rail are direct competitors because they have similar time and cost implications [29]. Punctuality in railways is of particular interest because railway systems are invariably managed by a higher authority who can control their operations more effectively. Delay in a railway context is typically defined as the difference between *scheduled* and *actual* arrival/departure times for a train at a halting station.

Delays in railway systems have potentially large economic costs. In Britain, a study estimated that their impact could be as high as GBP 0.54 per minute for passengers [23]. In the US, a similar study reported a delay cost of USD 232 per hour for U.S. Class 1 railroads (primarily freight), without accounting secondary effects such as revenue loss [27]. To some extent, *primary* delays (caused by equipment failures) are unavoidable. However, one can aim to reduce the *secondary* delays caused by knock-on effects during rescheduling, following an instance of primary delay [14].

Most railway systems operate with reference to a *timetable*, which is a conflict-free ideal operating schedule for all trains in the system. Dispatchers observe the operations in real-time, and are tasked with rescheduling trains whenever deviations from the timetable render the planned operations infeasible. Each dispatcher is typically responsible for a unique portion of the network (a string of stations on a single railway line, or a smaller number of stations around a junction where two or more railway lines meet) [25]. Given the short time available, limited human cognitive ability, and the arbitrary nature of particular delayed states, dispatchers make rescheduling decisions using thumb rules or heuristics.

Automated rescheduling (or real-time scheduling) systems aim to encode such heuristics into a fixed set of rules. We cover these methods in Section 2. Since the railway scheduling problem can be formulated in a mixed-integer linear programming framework, one can imagine solving it using exact techniques. However, the short reaction time constraints and the NP-hard nature of the formulation [2, 4] make this an infeasible proposition in the real world. On-line search techniques face a similar challenge [20, 32], since the search spaces are large and time is short. They tend to be applied to timetable generation rather than real-time application. Recently, approaches using Reinforcement Learning (RL) have shown better solution quality than heuristics [16]. RL is a feasible approach because the computational effort of training is expended off-line, while the on-line implementation is lightweight. We assume a similar viewpoint, with the difference that we employ policy search. In particular, we represent a key part of our scheduling policy as a neural network, and optimise its weights using the covariance matrix adaptation evolution strategy (CMA-ES) [11]. Several reasons support the use of a policy search approach.

(1) Policy search methods are attractive because they directly optimise the objective function (priority-weighted aggregate train delay), rather than a proxy as typically used in value-based RL [16].
(2) Since the number of states and actions in a railway network grow exponentially with the number of trains, realistic schedulers are constrained to make decisions based on only a small amount of "local" information. In the resulting non-Markovian environment, policy search is known to have an edge over value-based RL [9, 33].
(3) Policy search methods allow for encoding various forms of domain knowledge in a natural way (as described in Section 3). We open up only a small number of parameters for optimisation, thereby ensuring efficient search.
(4) Like RL, policy search, too, produces an associative mapping from state to action that can be used on-line in real-time. The computational effort it demands is primarily off-line. With the speed-up offered by parallelisation, the largest real-world test case in Section 5 requires 15 minutes per generation on 50 machines, and only a couple of hours in total.
(5) Most importantly, we obtain significantly better solutions (with lower delays) using policy search; Figure 1 provides a visual comparison with RL on three real lines in the Indian network. These sheer gains make policy search a method of choice for railway scheduling.

Since the value-based RL approach of Khadilkar [16] represents the state-of-the-art (our own experiments demonstrate that it outperforms several preceding approaches), we present our solution specifically as a comparison with RL. We use the same state and action space definitions as those used by Khadilkar [16]. However, instead of deriving an indirect policy from a value function using tabular Q-learning, we directly search in the space of policies using CMA-ES.

We begin with a review of related work on railway scheduling (Section 2) before formalising the task in Section 3. We describe our solution in Section 4 and present results in Section 5. We conclude with a discussion in Section 6.

2 RELATED WORK

A review of literature in the field of railway scheduling [1, 7] shows four broad approaches to solving the problem. First, there are studies that map the problem to job shop scheduling [4], and propose a mixed-integer linear programming (MILP) solution. They model arrival and departure times as continuous decision variables, and use binary indicator variables for resource allocation [22]. Related ideas include preprocessing for speed-up [8] and constraint programming [24]. While optimisation methods are feasible for developing reference timetables (a one-time exercise), they are not useful for real-time application because on-line MILP solvers have high computational complexity and slow response.

A second popular approach is known as Alternative Graphs (AG) [6, 19, 26], originally used for the no-store (or blocking) job shop scheduling problem. No-store means that a job cannot leave a machine (railway track) until the subsequent machine (track) becomes available. AG methods are typically focused on detecting and avoiding conflicts, such as when more than one train acquires a railway track at the same time. However, the computation of solutions has the same challenges with mathematical complexity as exact solutions.

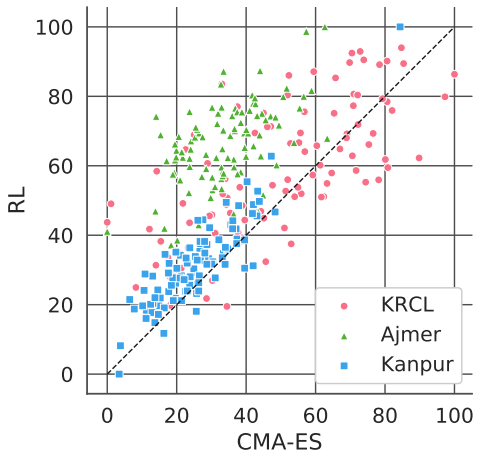

Figure 1: Normalised priority-weighted departure delay obtained by CMA-ES and RL on 100 perturbed timetables for three railway lines: KRCL, Ajmer, and Kanpur. Detailed results are in Section 5.

Third, many practical implementations use heuristics specially designed for railway scheduling (and rescheduling). Travel advance heuristic (TAH) [30] is a common method which solves the problem of scheduling by moving one train at a time. The performance of TAH is dependent on the order in which vehicles are moved, and the number of tracks used for 'look-ahead'. For every train, TAH sets the arrival time at the next node. If due to some conflicts, a train cannot move, it backtracks by requesting a delayed departure time at the current node. To reduce the amount of backtracks, Khadilkar [15] proposes a conflict-free logic for choosing the order of train movements. If the initial state of the system satisfies certain conditions, it proves that the scheduling can be completed without encountering any deadlock or backtracking. While feasible for use in real-time applications, handcrafted heuristics have unknown optimality gaps, and are not able to adapt to the nuances of specific problem instances.

Finally, recent studies have proposed the use of reinforcement learning (RL) for the real-time scheduling problem. Šemrov et al. [28] use Q-learning to learn a policy for railway scheduling. The state representation is comprehensive, and includes the current locations of the trains, current availability of each section, and time. The size of the action space is equal to the number of tracks in the network infrastructure. As the size of the state space is directly proportional to the railway network, the state space becomes unmanageable for realistic networks. To circumvent this combinatorial growth, Khadilkar [16] defines the state space of a train in terms of its *local* resources: a fixed number of resources behind and in front of the train. Since this number is independent of the actual network, it can easily scale up and transfer.

Evolutionary algorithms have been applied to many problems in transportation; we point out some examples. Cevallos and Zhao [3] present a genetic algorithm to minimise transfer time in bus transit, while Dundar and Sahin [5] present one for conflict resolution during re-scheduling. Huang et al. [13] evolve a binary encoding to optimise two objectives—energy consumption and travel time—on the Beijing-Yizhuang subway line. Similarly, Zhang et al. [34] propose a multi-objective particle swarm optimization algorithm to reduce both train delay and the number of trains delayed and test it on Beijing-Shanghai express railway.

3 TASK SPECIFICATION

A number of real-time scheduling algorithms are described in Section 4. All the algorithms run in conjunction with the discrete event simulation logic described in this section.

3.1 Simulating train movement on railway lines

In this work, we focus on the problem of scheduling railway *lines*, which implies that there are no junctions or branches in the simulation. However, trains can begin and/or end their journeys at any station, as long as they move in a single direction (either left-to-right or right-to-left as shown in Figure 2). The initial state can be an empty line (no trains pre-existing) or a predefined state (current locations and directions of trains).

Infrastructure. The physical infrastructure of the railway line is defined first. This consists of two types of resources: *stations* where trains can halt, and *sections* on which trains move from one station

to the next. Each station or section (*resource*) contains one or more parallel tracks, each of which can be occupied by at most one train at any time. For simplicity, we assume that any train is allowed to occupy any track in a resource. The number of tracks in a resource (hence the number of trains that the resource can hold simultaneously) is its *capacity*. The example in Figure 2 has 5 stations and 4 sections. There are 8 trains shown, 4 at each of the terminal stations. Trains starting at Alpha move towards the right, and vice versa. Each station on the line contains four parallel tracks, while the sections have a single track each.

Schedule constraints. Each simulation episode begins with a predefined initial state. For each train, we assume that there exists an ideal timetable that defines its desired arrival and departure times for each station on its journey (departure time from a station is equal to the arrival time in the next section. Additionally, we assume that the minimum traversal time on each section, the minimum halt time at each station, and the priority level are externally defined for each train. The two former quantities define constraints on the schedule. We discuss "priority level" later in this section.

Running the simulator. The overall flow of control in our simulator is shown in Figure 3. The simulator retains a list of upcoming events, one for each train. Each event corresponds to the time at which the next decision for the train will be needed. If the train has yet to start its journey, this will be the desired arrival time at its first station; if the train is at a station, it is the earliest feasible departure time from that station; and if the train is running in a section, it is the earliest feasible arrival time at the next station. The 'earliest feasible' time is the maximum of the ideal (timetable) time and the value computed by adding the minimum halt (or traversal) duration to the last known arrival (or departure). This ensures that minimum halt time and section traversal time constraints are respected, and that trains do not arrive or depart earlier than their scheduled times.

The simulation clock is set to the earliest time in the list of events. At each step, the set of trains with events at the current clock time is retrieved. If there is more than one train in the set, they are first sorted according to the residual capacities (free tracks) in their current resources, followed by their priorities. The sorting order is based on a deadlock-avoidance criterion proposed by Khadilkar [[2017]]. Then the trains are processed sequentially according to the logic described in Section 3.2. For trains that choose to wait in the current resource, the event time is incremented by 1 unit. For trains that choose to move to the next resource, the event times are updated according to the minimum halt/traversal durations.

A constraint known as *headway* (minimum time gap between departure of one train from a resource and arrival of next train into the same resource) is implemented by assuming that each train occupies one track on both resources (current and new) for 1 time unit[1]. When a train completes its journey (departure from last station on its route), it is removed from the event list. The simulation episode thus ends when all trains complete their journeys and the event list is empty. Should two or more trains get deadlocked (no further moves possible), the simulation terminates on crossing a predefined time threshold.

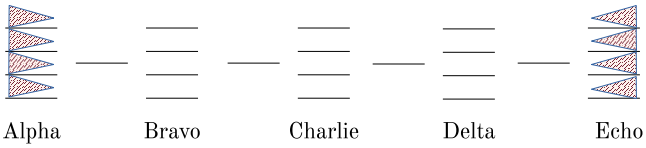

Figure 2: Illustrative railway line [16].

3.2 Logic for individual train decisions

The simulation logic is identical for all algorithms in this study, except for the shaded box in Figure 3. This box computes a binary decision for each train: whether to *move* to the next resource, or *wait* in the current resource. The logic is composed of two subroutines, described below.

Step 1: Preprocessing for deadlock avoidance. Since trains cannot move backwards except in very rare instances, a significant risk in railway lines is the possibility of deadlock [15, 18]. This condition occurs when trains heading in opposite directions occupy tracks on neighbouring resources in such a way that none of the trains can move forward. The station capacity is invariably higher than the section capacity, with the result that sections form the bottlenecks. Therefore, we introduce a simple heuristic that ensures that any train moving into a bottleneck (section) has a feasible future move (into the next station). We look at the next station on a train's journey, which is assumed to contain N_r tracks. If the next station (i) already holds N_r trains, or (ii) currently holds more than $(N_r - 2)$ trains heading in the same direction, a decision of *wait* is forced. Figure 4 shows four illustrative scenarios.

This logic is based on the concept of 'legal states' [18], which ensure that two sets of trains heading in opposite directions have a feasible set of moves to completely pass each other. The fact that we only look ahead to one station does not guarantee deadlock avoidance, but it ensures that deadlocks are sufficiently rare, so learning will mostly encounter useful policies. It can be shown that a stronger assumption such as reservation of at least one track at each station for unidirectional movement (one track for left-to-right only, another

[1] In reality, the headway constraint is imposed by the fact that trains move carriage-by-carriage onto the next track.

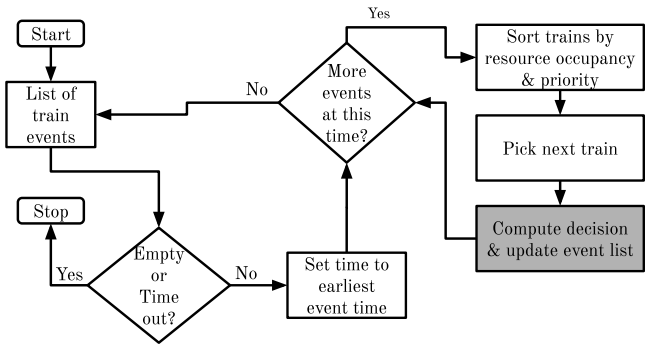

Figure 3: Flowchart of simulation logic. The shaded box is explained in Section 3.2, and the remaining boxes in Section 3.1.

for right-to-left only, and additional tracks available for both directions) always results in 'legal states', and is therefore guaranteed deadlock-free. However, this makes the schedules more conservative, and can result in greater values of delay. We include this logic in the comparisons in Section 5.

Step 2: Computing move/wait decisions. If the preprocessing logic in Step 1 is cleared, we refer to an optimised policy (Section 4) to decide whether the train should move to the next resource at this time, or wait in the current resource for one time step. The policy is represented by a neural network, with the states and actions as defined by Khadilkar [[2019]]. For each train, the state is formed by concatenating its priority code (externally defined) and a vector indicating the occupancy of a fixed number of resources (also externally defined) in the local neighbourhood. We assume that a total of l_b resources behind the train (opposite to its direction of movement) and l_f resources in front (in direction of movement) are included, along with the currently occupied resource (Figure 5). This formulation results in a state vector of length $(l_b + l_f + 2)$, consisting of $(l_b + l_f + 1)$ elements for resources and one element for the priority code. Throughout this work, we use $l_b = 2$ and $l_f = 6$; we find no significant improvement by varying these parameters.

The numerical value of resource state, as provided to the decision-making policy, is designed to indicate the availability (or otherwise)

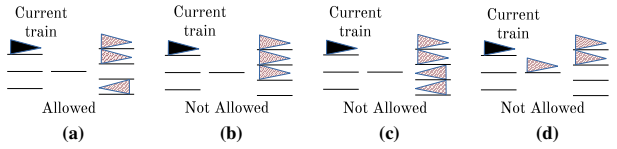

Figure 4: Preprocessing for avoiding deadlocks. Two stations with a single track section in between are shown. A solid triangle indicates the train for which a decision is to be taken, and the tapering side of the triangles indicate direction of movement. Moving in scenario (d) is not allowed because no track is available in the next resource.

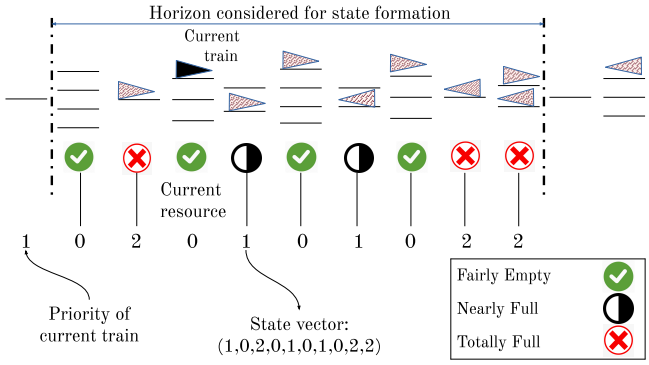

Figure 5: State vector formation for a train using local resources, with $l_b = 2$, $l_f = 6$, $w_c = w_d = 1$, $R = 3$. Values increase with occupancy, with 0 being 'fairly empty' and 2 being 'totally full'.

of free tracks in that resource. For each resource r, status S_r takes one of the R values from $\{0, 1, 2, 3, \ldots, R-1\}$. A larger value of S_r indicates greater congestion (fewer free tracks) in that resource. Let the number of tracks in a resource r be N_r, the number of tracks occupied by trains converging with the current train (moving in the opposite direction) be $T_{r,c}$, and the number of tracks occupied by trains diverging from the current train (moving in the same direction) be $T_{r,d}$. As per the assumption stated before, only one train can occupy a track at a time: hence, $T_{r,c} + T_{r,d} \leq N_r$. the status S_r is defined as follows.

$$S_r = R - 1 - \min(R-1, \lfloor N_r - w_c T_{r,c} - w_d T_{r,d} \rfloor),$$

where w_c and w_d are externally-defined weights. Figure 5 shows an example for mapping occupancy to status value of local resources, keeping $w_c = w_d = 1$ for simplicity of illustration. In the experiments in Section 5, we use $w_c = 0.9$ and $w_d = 1$. The de-emphasis on converging trains ensures that the algorithm prefers to pass trains traveling in the opposite direction (a requirement for schedule completion) as compared to trains moving in the same direction (which may lead to deadlock [18]).

Note that with a fixed size of the state vector, a finite number of train priority levels, and the predefined resource status levels R, the state space of the present formulation is finite and invariant with the length of the railway line, as well as the number of trains to be scheduled. As mentioned before, the action corresponding to any given state is binary: $\{0, 1\}$ or {wait, move}. Once a train decides to move, the actual track assignment is made according to a first-feasible logic. For trains heading right in Figure 5, we assign the first free track looking from the top down. For trains heading left, we assign the first free track when searched from the bottom up.

3.3 Objective function

The objective to be minimised is the summation of the train-priority-weighted delay of departure time for each train at each station in its journey. This definition has been previously used in the literature [5, 16]. A delay is defined as the difference between the scheduled departure time by the algorithm and the desired departure time in the initial timetable, with a lower bound of 0 (non-negative). It is formally defined as

$$J = \frac{1}{N_{dep}} \sum_{r,t} \frac{\delta_{r,t}}{p_t}, \tag{1}$$

where N_{dep} is the total number of departures in the timetable, $\delta_{r,t}$ is the delay for train t at resource (or station) r, and $p_t \in \{1, 2, \ldots, P\}$ is the priority of train t (note that $p_t = 1$ contributes the most, and has the greatest priority).

3.4 Benchmark railway lines

An initial timetable and the infrastructure of the railway line is required to set up the simulator. The initial timetable contains information about the journey of each train—stations, arrival time, departure time, priority, minimum halt time at each station for each train, and minimum traversal time for each train on each section. These times can be arbitrarily defined, based on the type of locomotive attached to the train, the number of carriages being pulled, gradation and turns on the route, and the priority of the train. The

infrastructure data contains all the stations, the number of tracks at a station, and the number of tracks in sections between two stations.

We evaluate our method on the same railway line data sets used by Khadilkar [[2019]] to benchmark his RL approach. The data set comprises two synthetic (hypothetical) instances (HYP-2, HYP-3) of increasing complexity, as listed in Table 1. The number of events processed by the simulator includes one event (time of entry) for each train at each resource on its route (stations and sections). In addition, the data set contains three real lines on the Indian Railway network. These are (i) 3 days of scheduled operations in 2014 on Roha-Tokur in western India, referred to as KRCL in this work, (ii) 3 days of scheduled operation in 2014 on Kanpur-Tundla in northern India, referred to as Kanpur, and (iii) 7 days of scheduled operation in 2014 on Ajmer-Palanpur in northwestern India, referred to as Ajmer. Note that the last two instances include sections with more than 1 track between stations, while the first three instances only contain single (bidirectional) tracks.

4 TRAINING WITH POLICY SEARCH

In this section, we describe our main contribution: the policy representation and optimisation steps that underpin the binary move/wait decisions for each train. The policy is represented by a neural network. With parameter values $l_b = 2$ and $l_f = 6$ as described earlier, the input size is $l_b + l_f + 2 = 10$. This is followed by three fully connected hidden layers of 10 neurons each, with *tanh* activation. The output is a softmax distribution over the two actions, and is therefore of size 2. The parameters of the network—totally 352 in number—are optimised in two steps, as described below.

4.1 Initialisation using imitation learning

Khadilkar [[2019]] suggests a set of conditions for finding a good initial policy, which can speed up training. As shown in Table 2, a 'move' probability is associated with states based on different conditions that they satisfy. States that do not satisfy any of the conditions are initialised to a probability of 0.5 for both actions. Since we represent the policy using a neural network rather than the tabular version used by Khadilkar [[2019]], we perform supervised learning to initialise the network parameters. The weights are trained to minimise *mean squared error* loss with respect to the values in Table 2, aggregated unweighted over all (59,049) possible state vectors. This initialisation allows us to set policy parameters that can complete a large fraction of schedules without deadlock. We then use CMA-ES to optimise the parameters further.

Scenario	Stations	Trains	Events	Timetable span
HYP-2	11	60	1320	4 hours
HYP-3	11	120	2640	7 hours
KRCL	59	85	5418	3 days
Kanpur	27	190	7716	3 days
Ajmer	52	444	26258	7 days

Table 1: Summary of railway line scenario scales.

4.2 Using CMA-ES for policy search

CMA-ES [10] is a policy search method that randomly generates a population of solutions first around an initial seed, and thereafter moves the generating distribution in a direction maximising fitness. We pick this method based on its impressive track record on tasks as diverse as network security [12], hydro-engineering [31], and robot soccer [17].

While the initial guess provided is often zero or random, we seed CMA-ES with the imitation-learned network parameters. All instances in this study use the same initial guess for mean parameter values. We observe that starting CMA-ES with imitation-learned weights always resulted in faster convergence than using other initialisation techniques.

We minimise the fitness function given in (1). The population size is kept at 51 for all experiments, and the fitness value of each individual (a stochastic policy) is the average of 10 simulation runs. Training is stopped when the population mean converges (remained within a small neighbourhood over several generations). The best-performing individual in each of the last 50 generations is chosen as the winner. Training was done in Python on a cluster of machines, each with an Intel Core i5-4690 CPU @ 3.50GHz with 2 cores (4 threads) and 8 GB of RAM.

5 RESULTS

From Figure 6, we observe that CMA-ES steadily reduces the objective function and has low variance across generations when training is terminated.

Table 3 compares the priority-weighted delays for several competing algorithms. The reported numbers are averaged over the same set of 100 independent timetables (none of these were used for training) for all algorithms. The test timetables were produced by random perturbations to the departure times of trains, drawn from a uniform random distribution over $[-30, 30]$ units, and added to the original defined (or obtained) timetables. The algorithms being compared are CMA-ES (proposed approach), RL (Q-learning, [16]), TAH-FP (fixed priority, [30]) and TAH-CF (critical first, [15]). Results from three baselines are also shown: (i) a naive greedy logic which moves the train to next resource whenever a track is free in the immediate

#	States Satisfying	Move Probability
1	Next resource is full	0.0
2	At least 3 consecutive resources are full	0.10
3	Next resource is almost full, next-but-one is full	0.15
4	Average status of upcoming resources is between 0.5 and 1	0.85
5	Average status of upcoming resources is less than 0.25	0.95

Table 2: Initial conditions learned using supervised learning (adapted from Khadilkar [[2019]]). Trained neural network's weights were used as a starting distribution mean for CMA-ES.

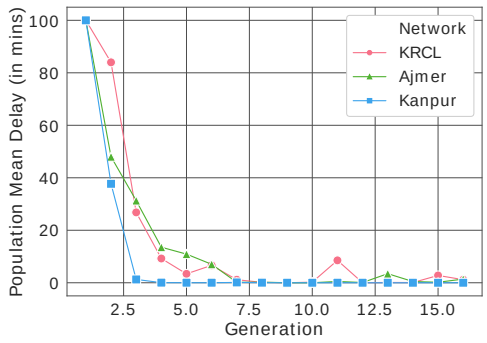

Figure 6: Normalised priority-weighted departure delay (in mins) for each generation for KRCL, Ajmer and Kanpur railway lines

next resource, (ii) a version of greedy that uses the preprocessing logic from Section 3, reserving one track for each direction when possible, but always chooses to *move* when Step 1 is cleared, and (iii) path-to-destination [18] where a train is moved only if there is a free track in all resources ahead of it until its final destination.

We note that CMA-ES results in the lowest average priority-weighted delays among all algorithms, for all 5 scenarios. In addition to reporting the average delay, we report two performance metrics in Table 3. The first are the numbers in parentheses, which list the number of instances that resulted in deadlock (failure to complete schedule). All algorithms except for CMA-ES, RL, and PTD (which is guaranteed deadlock-free but very conservative) result in deadlocks for at least a few instances. Average delays reported for these algorithms exclude the deadlocked instances. Finally, we also report the number of instances where CMA-ES resulted in a lower delay than RL. We observe that as the traffic density increases (scenarios with more trains on fewer stations in the same amount of time, as given in Table 1), CMA-ES possesses a greater advantage.

6 CONCLUSION

The goal of this paper was to explore evolutionary search to optimise policy parameters for on-line scheduling in railways. We showed that our formulation, coupled with a specific policy search method (CMA-ES), was able to outperform previously proposed methods including RL (in tabular Q-learning form) and established heuristics. We restricted our claims and comparisons with respect to these methods because (i) the on-line nature of the problem puts a limit on the maximum response time (not achievable using on-line search or exact methods), and (ii) the scale of the problem instances has not been successfully addressed by exact methods in literature. Given the off-line-intensive, on-line-lightweight nature of our proposed method, we believe that it could be used effectively in real-world railway lines.

In future work, we would like to extend the method to handle full railway networks with branches (as opposed to the linear topology considered in this work). This will require an extension to the model, but the problem space is already of interest to researchers [21]. An equally significant research question revolves around the use of explicit coordination between trains (agents) when moving through resources, which could lead to more efficient solutions.

REFERENCES

[1] V. Cacchiani, D. Huisman, M. Kidd, L. Kroon, P. Toth, L. Veelenturf, and J. Wagenaar. 2014. An overview of recovery models and algorithms for real-time railway rescheduling. *Trans. Res. Part B: Methodological* 63 (2014), 15–37.

[2] X Cai and C Goh. 1994. A fast heuristic for the train scheduling problem. *Computers & OR* 21, 5 (1994), 499–510.

[3] F. Cevallos and F. Zhao. 2006. Minimizing transfer times in public transit network with genetic algorithm. *Transportation Research Record* 1971 (2006), 74–79.

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

[5] S. Dundar and I. Sahin. 2013. Train re-scheduling with genetic algorithms and artificial neural networks for single-track railways. *Trans. Res. Part C: Emerging Technologies* 27 (2013), 1–15.

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

[7] Wei Fang, Shengxiang Yang, and Xin Yao. 2015. A survey on problem models and solution approaches to rescheduling in railway networks. *IEEE Transactions on Intelligent Transportation Systems* 16, 6 (2015), 2997–3016.

[8] M. Fischetti and M. Monaci. 2017. Using a general-purpose Mixed-Integer Linear Programming solver for the practical solution of real-time train rescheduling. *European Journal of OR* 263, 1 (2017), 258–264.

[9] Faustino J. Gomez and Risto Miikkulainen. 1999. Solving Non-Markovian Control Tasks with Neuro-Evolution. In *Proc. IJCAI 1999*. Morgan Kaufmann, 1356–1361.

[10] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. 2019. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634. (Feb. 2019). https://doi.org/10.5281/zenodo.2559634

[11] N. Hansen and A. Auger. 2011. CMA-ES: Evolution Strategies and Covariance Matrix Adaptation. In *Annual Conference on Genetic and Evolutionary Computation*. 991–1010.

[12] Guan-Yu Hu, Zhi-Jie Zhou, Bang-Cheng Zhang, Xiao-Jing Yin, Zhi Gao, and Zhi-Guo Zhou. 2016. A method for predicting the network security situation based on hidden BRB model and revised CMA-ES algorithm. *Applied Soft Computing* 48 (2016), 404 – 418. https://doi.org/10.1016/j.asoc.2016.05.046

[13] Y. Huang, L. Yang, T. Tang, F. Cao, and Z. Gao. 2016. Saving Energy and Improving Service Quality: Bicriteria Train Scheduling in Urban Rail Transit Systems. *IEEE Trans. on ITS* 17, 12 (Dec 2016), 3364–3379.

[14] H. Khadilkar. 2016. Data-enabled stochastic modeling for evaluating schedule robustness of railway networks. *Transportation Science* 51, 4 (2016), 1161–1176.

[15] H. Khadilkar. 2017. Scheduling of vehicle movement in resource-constrained transportation networks using a capacity-aware heuristic. *Amer. Control Conf.* (2017), 5617–5622.

	CMAES	Win/Lose/Tie	RL	TAH-FP	TAH-CF	Naive greedy	Greedy with preproc.	PTD
HYP-2	**4.28 (0)**	91 / 9 / 0	4.78 (0)	4.58 (0)	5.93 (0)	11.16 (2)	4.35 (0)	714.00 (0)
HYP-3	**15.50 (0)**	100 / 0 / 0	18.54 (0)	61.89 (97)	140.14 (95)	- (100)	16.35 (0)	2003.98 (0)
KRCL	**42.34 (0)**	66 / 34 / 0	43.04 (0)	46.41 (8)	47.02 (0)	- (100)	42.40 (0)	4714.08 (0)
Ajmer	**3.92 (0)**	100 / 0 / 0	4.65 (0)	10.76 (3)	5.99 (0)	9.25 (76)	3.99 (3)	8304.84 (0)
Kanpur	**1.54 (0)**	87 / 13 / 0	1.66 (0)	2.19 (0)	2.28 (0)	1.85 (0)	1.54 (0)	313.60 (0)

Table 3: Priority-weighted departure delay (in mins) averaged over 100 runs with perturbed versions of timetable used for training

[16] H. Khadilkar. 2019. A Scalable Reinforcement Learning Algorithm for Scheduling Railway Lines. *IEEE Trans. on ITS* 20, 2 (Feb 2019), 727–736.

[17] Patrick MacAlpine, Samuel Barrett, Daniel Urieli, Victor Vu, and Peter Stone. 2012. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press.

[18] S. Mackenzie. 2010. *Train scheduling on long haul railway corridors*. Ph.D. Dissertation. University of South Australia.

[19] A. Mascis and D. Pacciarelli. 2002. Job-shop scheduling with blocking and no-wait constraints. *European Journal of OR* 143, 3 (2002), 498–517.

[20] K. Nitisiri, M. Gen, and H. Ohwada. 2019. A parallel multi-objective genetic algorithm with learning based mutation for railway scheduling. *Computers & Industrial Engineering* 130 (2019), 381–394.

[21] Erik Nygren, Adrian Egli, Giacomo Spigler, Mattias Ljungström, Jeremy Watson, Christian Eichenberger, Guillaume Mollard, and Sharada Mohanty. 2019. Flatland Challenge: Multi Agent Reinforcement Learning on Trains. (07 2019). https://doi.org/10.13140/RG.2.2.24477.05601

[22] P. Pellegrini, G. Marliere, and J. Rodriguez. 2012. Real time railway traffic management modeling track-circuits. In *ATOMOS 2012*. France.

[23] J. Preston, G. Wall, R. Batley, J. Ibanez, and J. Shires. 2009. Impact of Delays on Passenger Train Services: Evidence from Great Britain. *Trans. Res. Record* 2117, 1 (2009), 14–23.

[24] J. Rodriguez. 2007. A constraint programming model for real-time train scheduling at junctions. *Trans. Res. Part B: Methodological* 41, 2 (2007), 231–245.

[25] E. Roth, N. Malsch, and J Multer. 2001. *Understanding how train dispatchers manage and control trains: results of a cognitive task analysis*. Technical Report. United States Federal Railroad Administration.

[26] M. Sama, A. D'Ariano, F. Corman, and D. Pacciarelli. 2017. A variable neighbourhood search for fast train scheduling and routing during disturbed railway traffic situations. *Computers & OR* 78 (2017), 480–499.

[27] B. Schlake, C. Barkan, and J. Edwards. 2011. Train Delay and Economic Impact of In-Service Failures of Railroad Rolling Stock. *Trans. Research Record* 2261, 1 (2011), 124–133.

[28] D. Šemrov, R. Marsetič, M. Žura, L. Todorovski, and A. Srdic. 2016. Reinforcement learning approach for train rescheduling on a single-track railway. *Trans. Res. Part B: Methodological* 86 (2016), 250–267.

[29] N. Shinghal. 2005. Rail-Road Competition in Freight Transportation: Price and Service Issues. *Economic and Political Weekly* 40, 25 (2005), 2587–2593.

[30] S. Sinha, S. Salsingikar, and S. SenGupta. 2016. An iterative bi-level hierarchical approach for train scheduling. *JRTPM* 6, 3 (2016), 183–199.

[31] Hassan Smaoui, Lahcen Zouhri, Sami Kaidi, and Erick Carlier. 2018. Combination of FEM and CMA-ES algorithm for transmissivity identification in aquifer systems. *Hydrological Processes* 32, 2 (2018), 264–277. https://doi.org/10.1002/hyp.11412 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/hyp.11412

[32] P. Tormos, A. Lova, F. Barber, L. Ingolotti, M. Abril, and M. Salido. 2008. A genetic algorithm for railway scheduling problems. In *Metaheuristics for scheduling in industrial and manufacturing applications*. Springer, 255–276.

[33] S. Whiteson and P. Stone. 2006. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research* 7, May (2006), 877–917.

[34] L. Zhang, Y. Qin, X. Meng, L. Wang, and T. Zhu. 2016. MPSO-Based Model of Train Operation Adjustment. *Procedia Engineering* 137 (2016), 114–123.