In Proc. ICC 2021, pp. 75--80, IEEE, 2021.

Optimising a Real-time Scheduler for Indian Railway Lines by Policy Search

Rohit Prasad, Harshad Khadilkar, and Shivaram Kalyanakrishnan

***Abstract*— The Indian railway network carries the largest number of passengers in the world, with over 8.4 billion transported in 2018, in addition to 1.2 billion tonnes of freight [1]. Nonetheless, the network has only about a tenth the "track-length per passenger" of the U.S., and half that of China [2]. This severe limitation of infrastructure, coupled with variability and heterogeneity in operations, raises significant challenges in scheduling. In this paper, we describe a policy search approach to decide arrival/departure times and track allocations for trains such that the resource and operating constraints of the railway line are satisfied, while the priority-weighted departure delay (PWDD) is minimised. We evaluate our approach on three large railway lines from the Indian network. We observe significant reductions of PWDD over traditional heuristics and a solution based on reinforcement learning.**

I. Introduction

The Indian railway network has been vital to the growth of the Indian economy, transporting many billions of passengers and tonnes of freight every year [1]. Yet, the operations of the network remain confounded by many factors. Foremost is the significant shortage of infrastructure—only about half the length of track per passenger as China's and a tenth that of the U.S. [2]. With such a high density of trains, delays can have severe cascading effects. The long distances of travel, the prevalence of non-periodic trains and multiple priority classes, discordant decision-making within sub-networks, all create the need for *real-time* replanning of schedules [3]. Static fixes such as "allowances" to particular trains often have unintended effects on the rest of the network [4].

Delay in a railway context is typically defined as the difference between *actual* and *scheduled* arrival/departure times for a train at a halting station. Delays have potentially large economic costs [5]. 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 following an instance of primary delay [3].

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 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 an intersection) [6]. Given the short time available and limited human cognitive ability, dispatchers usually make rescheduling decisions based on thumb rules or heuristics. This approach leaves significant room for improvement when the performance of the entire network is taken into account.

In principle, the railway scheduling problem can be formulated as a mixed-integer linear program, which one can imagine solving using exact techniques. However, short reaction times for replanning, coupled with the NP-hardness of the problem [7] render this option infeasible in practice. On-line search techniques face a similar challenge since the search spaces are large and planning time is short [8], [9]. Computationally intensive methods might be suitable for timetable generation, but are not for real-time application.

Recent approaches using Reinforcement Learning (RL) [10] have produced better solutions than traditional heuristics. RL is a feasible approach because the computational effort of training is expended off-line, while the on-line effort is lightweight. RL algorithms such as Q-learning are designed assuming Markovian dynamics over the state variables used for decision making. For an application such as railway scheduling, with hundreds of trains and resources, it is virtually impossible to represent full state information in a form that is amenable to generalisation and learning. Instead, it is typical for designers to construct "compact, yet useful" representations—whose efficacy is ultimately validated empirically.

In this paper, we propose a *policy search* (PS) approach to railway scheduling. PS may be viewed as a form of black box optimisation, wherein a search is performed over the parameters of a policy to maximise a specified objective function. In our solution, a key element of the scheduler is represented as a neural network, whose weights are treated as policy parameters. Several factors motivate the use of PS for railway scheduling.

1) PS methods directly optimise the objective function (priority-weighted departure delay in our case). In contrast, value-based RL methods such as Q-learning [10] approximate long-term utilities; even small errors can degrade performance on tasks with long horizons.
2) Since the number of states in a railway network grows exponentially with the number of trains, realistic schedulers are constrained to make decisions based on only a small amount of "local" information. It is well-known that PS can outperform value-based RL when the environment is non-Markovian [11].
3) PS methods allow for encoding various forms of domain knowledge in a natural way. We open up only a small number of parameters (≈ 300) for optimisation, thereby ensuring efficient search.
4) Like RL, the output of PS is also an associative mapping from state to action, which can be used on-line in

RP was with the Indian Institute of Technology Bombay while carrying out this work. HK is with TCS Research. SK (corresponding author, e-mail: shivaram@cse.iitb.ac.in) is with the Indian Institute of Technology Bombay.

real-time, taking only few milliseconds to generate a revised timetable. The bulk of the computational effort is in off-line training, where, too, PS can enjoy the benefit of parallelisation. Our largest real-world test case (see Section VI) requires about two hours of training, when run on 50 machines.

5) Most importantly, we obtain significantly better solutions (lower delays) using PS; Figure 1 shows a comparison with RL on three Indian railway lines. These gains are our strongest reason for proposing PS as the method of choice for railway scheduling.

As the RL approach of Khadilkar [10] 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 [10]. However, instead of indirectly deriving a policy from an action value function, we directly search the policy space using the "covariance matrix adaptation evolution strategy" (CMA-ES) [12], a popular algorithm for black box optimisation.

We review related work on railway scheduling (Section II) before formalising the task in Section III. In Section IV we describe our contribution based on domain knowledge to avoid deadlocks and to encode an effective policy template. We describe our PS solution in Section V, and present results in Section VI. We conclude with a discussion in Section VII.

II. Related Work

A review of literature on railway scheduling shows four broad approaches [13]. First, there are studies that map the problem to job shop scheduling [7], 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 [14]. While MILP solvers are feasible for developing reference timetables (a one-time exercise), they are not useful for real-time application because of their high computational complexity and slow response.

A second popular approach is that of Alternative Graphs (AG), originally used for the no-store (or blocking) job shop scheduling problem [15], [16]. However, the solutions using AG have the same challenges with mathematical complexity as exact techniques.

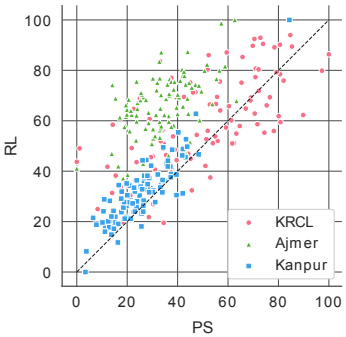

Fig. 1: Scatterplot of normalised priority-weighted departure delay obtained by PS and RL on 100 "perturbed" timetables for three railway lines: KRCL, Ajmer, and Kanpur. Detailed results and explanations are given in Section VI.

Third, many practical implementations use heuristics specifically designed for railway scheduling and rescheduling. The "travel advance heuristic" (TAH) [17] solves the problem of scheduling by moving one train at a time, and backtracking in case of conflicts. To reduce the amount of backtracking, Khadilkar [18] proposes a conflict-free logic for choosing the order of train movements. If the initial state satisfies certain conditions, it is shown that scheduling can be completed without encountering any deadlock or backtracking. While feasible for use in real-time applications, TAH is not able to adapt to the nuances of specific problem instances (see Section VI).

Finally, recent studies have proposed the use of reinforcement learning (RL) for the real-time scheduling problem. Šemrov et al. [19] use Q-learning to learn a policy for railway scheduling. However, the state space becomes unmanageably large for realistic networks. To circumvent this combinatorial growth, Khadilkar [10] 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. In this representation, the number of states is independent of the actual size of the railway network—convenient to scale up and transfer. Whereas Khadilkar [10] implements the tabular version of Q-learning, we apply PS with the same state representation.

PS approaches such as genetic algorithms have been applied to many problems in transportation, such as minimising transfer time in bus transit [20] and conflict resolution during re-scheduling [21]. Some formulations simultaneously optimise multiple objectives, such as energy and travel time [22].

III. Task Specification

Like Khadilkar [10], we focus on the problem of scheduling in railway *lines*, implying there are no junctions or branches in the train routes. However, trains can begin and 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). The input to our simulation is a specification of the infrastructure and schedule constraints.

Infrastructure. The physical infrastructure of a railway line 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

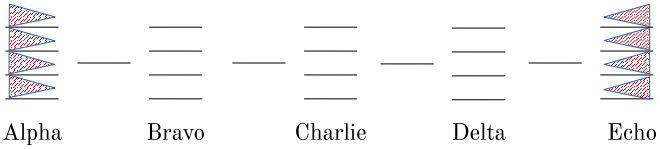

Fig. 2: Illustrative railway line [10] with 8 trains and 5 stations. Explanations are in the text.

at most one train at any time. For simplicity, we assume that any train is allowed to occupy any track in a resource. Manually operated lines typically designate separate tracks for the two directions of movement; however, this limits the solution space and could lead to suboptimal solutions. 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 those at Echo towards the left. Each station on the line contains four parallel tracks, while sections have a single track each.

Schedule constraints. Each simulation episode begins with a predefined initial state. For each train, we assume there exists an ideal timetable giving its desired arrival and departure times for each station on its journey (departure time from a *station* is equal to the arrival time at the next *section*). Additionally, we assume that for each train, we are given the minimum traversal time on each section, the minimum halt time at each station, and a "priority level". The two former quantities define constraints on the schedule, while priority feeds into the objective function.

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 time 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 definition 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

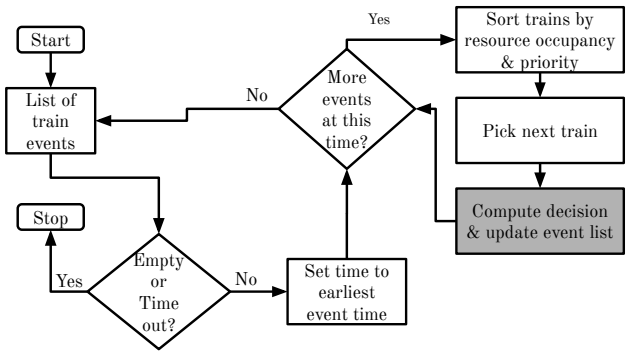

Fig. 3: Flowchart of simulation logic. The shaded box is explained in Section IV.

in the set, the trains 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 [18]. Then the trains are processed sequentially as per the shaded box in Figure 3 (described in Section IV). 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, updates are as per the minimum halt/traversal durations.

A constraint known as *headway* (minimum time gap between the departure of one train from a resource and the arrival of the 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 (a user-defined parameter). In reality, the headway constraint is imposed by the fact that trains move carriage-by-carriage onto the next track. When a train completes its journey (departure from last station on its route), it is removed from the event list. The episode ends when the event list is empty, or, should two or more trains get deadlocked (no further moves possible), a predefined time threshold is crossed.

Objective function. For a given scheduling task, the objective to be minimised is the average of the priority-weighted delay of departure time for each train at each station in its journey—"PWDD" in short. This definition has been previously used in the literature [10], [21]. 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. Formally,

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

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).

IV. Logic for individual train decisions

The simulation logic described in the previous section is identical for all the algorithms evaluated in this study: what distinguishes the algorithms is the shaded box in Figure 3. This module computes a binary decision for any given input train (as mentioned earlier, trains are chosen according to the logic of Khadilkar [18]): whether to *move* it to the next resource, or to *wait* in the current resource. In our proposed solution, the first step is a novel preprocessing scheme to minimise the incidence of deadlocks. This carefully-designed scheme is a helping hand based on our domain knowledge to ease the subsequent step of optimisation. We reuse the definition of state by Khadilkar [10] in the second step, but thereafter apply PS, rather than RL (see Section V).

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 [18], [23]. 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. Since station capacities are invariably higher than section capacities, sections usually form the bottlenecks. We introduce a simple heuristic to ensure that any train moving into a 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 (1) already holds N_r trains, or (2) holds more than $(N_r - 2)$ trains heading in the same direction, a decision of *wait* is *forced*. Figure 4 shows four illustrative scenarios related to deadlocks.

Our logic is based on the concept of "legal states", which ensure that two sets of trains heading in opposite directions have a feasible set of moves to completely pass each other [23]. Our look-ahead of only one station does not guarantee deadlock avoidance, but it ensures that deadlocks are sufficiently rare, so PS will mostly encounter useful policies. It can be shown that a stronger assumption (such as reserving one track for left-to-right traffic and another one for right-to-left at each station) will always result in legal states, thereby guaranteeing deadlock-free operation. However, this approach results in more conservative schedules, implying larger values of PWDD. We include this logic in our comparisons in Section VI.

Step 2: Computing *move/wait* decisions. If the preprocessing logic in Step 1 is cleared, we refer to an optimised policy (Section V) to decide whether the train should move to the next resource or wait in the current resource for one time step. Our policy is represented by a neural network, with the input (states) and output (actions) as defined by Khadilkar [10]. For each train, the state is formed by concatenating its given priority code and a vector indicating the occupancy of a fixed number of resources 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 (see Figure 5). This formulation results in a state vector consisting of $l_b + l_f + 1$ elements for resources and one for the priority. We use $l_b = 2$ and $l_f = 6$, as does Khadilkar [10], finding no significant improvement by varying these parameters.

The occupancy values in the state vector, as provided to the decision-making policy, indicate the availability (or otherwise) of free tracks in corresponding resources. For each resource r, status S_r takes one of R values from $\{0, 1, 2, \ldots, R-1\}$, indicating the number of free tracks. 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 stated earlier, we assume only one train can occupy a track at a time, implying $T_{r,c} + T_{r,d} \leq N_r$. The status S_r is defined as

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

where w_c and w_d are externally-defined weights. Note that R and S_r together define a status variable which takes only R unique integer values, resulting in a state space of fixed size. The value of R can be chosen based on the largest number of tracks that occurs *commonly* among the resources; any resources with an exceptionally large number of tracks are unlikely to be bottlenecks. Khadilkar [10], who uses tabular values, chooses R based on the acceptable number of unique states. If each resource takes R unique status values and there are $(l_b + l_f + 1)$ resources in the state vector, then the number of unique resource states is $R^{l_b+l_f+1}$, a finite and invariant number regardless of the size of railway line. Figure 5 shows an example of mapping occupancy to status value of local resources, keeping $w_c = w_d = 1$ for simplicity of illustration. Our implementation uses $w_c = 0.9$, $w_d = 1$, $R = 3$, the same values used by Khadilkar [10]. The de-emphasis on converging trains prompts algorithms to prefer passing trains traveling in the opposite direction (required for schedule completion) as compared to trains moving in the same direction (which may lead to deadlock [23]).

As mentioned earlier, the action corresponding to any given state is binary: one of {*wait*, *move*}. Once a train decides to move, the actual track assignment is made according to a first-feasible logic. For example, 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.

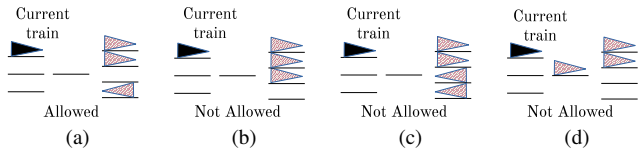

Fig. 4: Preprocessing for avoiding deadlocks. Two stations with a single track section in between are shown. A solid black triangle indicates the train for which a decision is to be taken, and the tapering portion of the triangles indicate direction of movement. Moving in scenario (d) is not allowed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

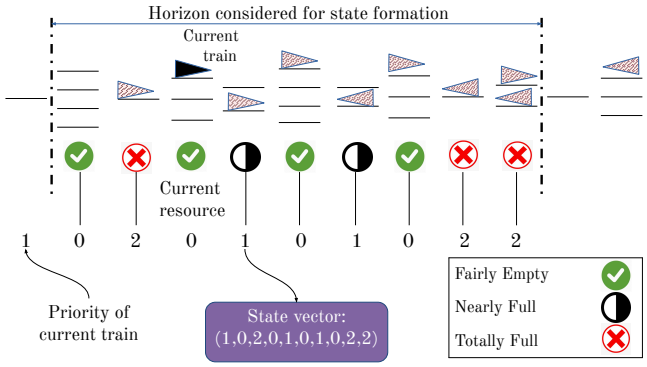

Fig. 5: State vector formation for a train using local resources, with $l_b = 2$, $l_f = 6$, $w_c = w_d = 1$, $R = 3$. Values increase with occupancy, with 0 being 'fairly empty' and 2 being 'totally full'.

V. Training with Policy Search

In this section, we describe our main innovation: the policy representation used for making *move*/*wait* decisions for each train, and its optimisation using PS. With the aim of realising a non-linear, yet compact mapping from states to action probabilities, we represent our policy as a neural network (see Figure 6). With parameter values $l_b = 2$ and $l_f = 6$ as described earlier, the input size is $l_b + l_f + 2 = 10$. Thereafter follow three fully connected hidden layers of 10 neurons each, with *tanh* activation. The output is a softmax distribution over the two actions. This architecture results in a total of 352 parameters to optimise.

In principle, we can use any form of local search (such as hill-climbing and simulated annealing) for optimising our policy weights. Rather than systematically assess the many available options, we go with the covariance matrix adaptation evolution strategy (CMA-ES) [12], which has yielded impressive results on tasks as diverse as hydro-engineering [24] and robot soccer [25]. Moreover, CMA-ES has ready implementations and effective default parameters (such as selection ratios and step sizes, which we leave unchanged). CMA-ES is an evolutionary algorithm that randomly generates a population of solutions around an initial seed, then moves the generating distribution in a direction maximising fitness. We initialise it with the zero vector and a standard deviation of 0.5 along each dimension.

Whereas Q-learning [10] performs learning updates based on each state-to-next-state transition, our proposed PS approach considers only a single feedback signal per policy: PWDD given in (1) aggregated at the end of each episode across all trains. We believe that this reduces the noise in feedback produced by tracking delays for individual trains, since states are only partially observable with the "local" representation. The population size is 51 for all experiments, and the fitness of each individual (a stochastic policy) is the average of 10 simulation runs. Training is stopped when the population mean remains within a small neighbourhood over several generations. The best-performing individual from the last 50 generations is chosen winner. Training is done on a 50-node cluster of Intel Core i5-4690/3.50GHz CPU machines, each with 2 cores and 8 GB of RAM.

VI. Results

We evaluate our PS method on the same railway lines used by Khadilkar [10] to benchmark his RL approach. Recall from Section III that a task specification comprises (1) infrastructure data describing the complete topology of the network (stations, sections, tracks), and (2) schedule constraints on the journey of each train—stations, arrival time, departure time, priority, minimum halt time at each station, and minimum traversal time on each section. The data set contains two synthetic instances—SYN-1 and SYN-2 (denoted "HYP-2" and "HYP-3" by Khadilkar [10])—as well as three real lines from the Indian railway network. The latter are (1) 3 days of scheduled operations in 2014 on Roha-Tokur in western India, referred to as KRCL, (2) 3 days of scheduled operation in 2014 on Kanpur-Tundla in northern India, referred to as Kanpur, and (3) 7 days of scheduled operation in 2014 on Ajmer-Palanpur in northwestern India, referred to as Ajmer. The last two instances include sections with more than one track between stations, while the first three instances only contain single (bidirectional) tracks. The real-world instances are of extended duration because disturbances in the timetable propagate for a long time in congested lines. We wish to investigate whether the schedule is stable (the line returns to 0-delay operation in a finite amount of time) or unstable.

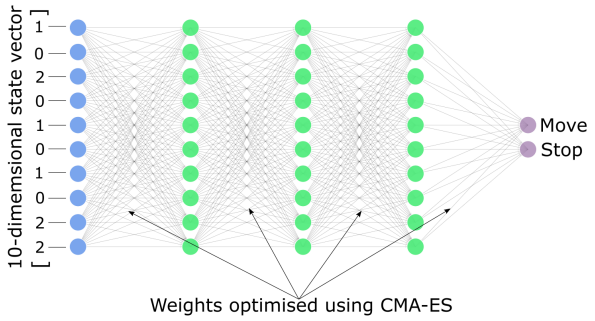

Fig. 6: Policy as a 3-hidden-layer neural network.

Table I summarises this data set and also provides our results. The number of "events" includes one for each train at each resource (stations and sections) on its route. We verify that our PS method (based on CMA-ES) steadily decreases the objective function (PWDD) over generations. To ensure that the method does not "overfit" to the given timetable, we test it on 100 "perturbed" timetables, created by adding noise drawn uniformly at random from $[-30, 30]$ minutes to the original timetables. The same 100 timetables (not used for training) are used to evaluate all the algorithms. We compare PS with the Q-learning (RL) algorithm of Khadilkar [10], as well as the "fixed-priority" and "criticial first" variants of the travel advance heuristic described in Section II. We denote these variants TAH-FP [17] and TAH-CF [18], respectively. Table I compares the PWDD values of these algorithms. Also included in the table are three other baselines: (1) a naïve greedy approach that moves the train whenever a track is free in the next resource; (2) a version of the previous method that additionally uses the preprocessing logic from Section III, reserving one track for each direction when possible, but always choosing to *move* when Step 1 is cleared; and (3) the "path-to-destination" (PTD) strategy [23], under which a train is moved only if there is a free track in *all* the resources ahead of it until its final destination.

Foremost, we observe that PS yields the lowest PWDD values among all algorithms and lines. All algorithms except for PS, RL, and PTD (which is guaranteed deadlock-free) result in deadlocks for at least a few instances (average delays reported for these algorithms exclude the deadlocked instances). The table also reports the number of instances on which PS resulted in a lower delay than RL. We notice that as the traffic density increases, PS appears to offer a greater advantage. This trend is also visible in Figure 1.

Line	Stations	Trains	Events	Span	PS	Win/Lose/Tie	RL	TAH-FP	TAH-CF	Naïve greedy	Greedy with preproc.	PTD
SYN-1	11	60	1320	4 hours	**4.28 (0)**	91 / 9 / 0	4.78 (0)	4.58 (0)	5.93 (0)	11.16 (2)	4.35 (0)	714.00 (0)
SYN-2	11	120	2640	7 hours	**15.50 (0)**	100 / 0 / 0	18.54 (0)	61.89 (97)	140.14 (95)	- (100)	16.35 (0)	2003.98 (0)
KRCL	59	85	5418	3 days	**42.34 (0)**	66 / 34 / 0	43.04 (0)	46.41 (8)	47.02 (0)	- (100)	42.40 (0)	4714.08 (0)
Ajmer	52	444	26258	7 days	**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	27	190	7716	3 days	**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 I: For each of five railway lines, columns 2–5 give properties of the lines; the remaining columns (except 7) show PWDD (in minutes) averaged over 100 perturbed versions of the timetable used for training. Accompanying in parentheses is the number of deadlocks resulting from these runs. The win/lose/tie data in column 7 is for PS vs. RL.

VII. Conclusion

We propose a PS approach to optimise a real-time scheduler for railway lines. Our founding hypothesis—that such an approach could outperform a recent one based on RL [10], as well as established heuristics—is validated on three large lines in the Indian railway network. Offering significant reductions in delay, while still being able to recompute schedules in real time, our approach bears promise to assist (and gradually decrease) human decision making in the systems currently deployed on these lines. Indeed the KRCL, Kanpur, and Ajmer data sets were sourced directly from Indian railway authorities, so we may expect the divergence between simulation and reality to be relatively minor. The next step would be to pilot our approach on an actual railway line, which must begin with instrumentation to provide state information as a *live feed*.

Acknowledgments

The authors are grateful to Shripad Salsingikar from TCS for helping procure the real-world data sets used in this paper, and to Abhiram Ranade from IIT Bombay for providing useful insights on our algorithm. Shivaram Kalyanakrishnan was partially supported by SERB grant ECR/2017/002479.

References

[1] Ministry of Railways. Indian railways year book 2018–19. Technical report, Government of India, 2019. URL https://www.indianrailways.gov.in/railwayboard/uploads/directorate/stat_econ/Year_Book/Year%20Book%202018-19-English.pdf.

[2] International Union of Railways. Railway statistics synopsis 2019. Technical report, 2019. URL https://web.archive.org/web/20190805180138/https://uic.org/IMG/pdf/uic-statistics-synopsis-2019.pdf.

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

[4] Narayan Rangaraj and Madhu Belur. A concept note for railway timetabling to rationalize and improve capacity utilization. Technical report, 2018. URL http://niti.gov.in/sites/default/files/2019-01/Report%20on%20A%20concept%20note%20for%20railway%20timetabling%20to%20rationalize%20and%20improve%20capacity%20utilization_0.pdf.

[5] J. Preston, G. Wall, R. Batley, J. N. Ibañez, and J. Shires. Impact of delays on passenger train services: Evidence from Great Britain. *Trans. Res. Record*, 2117(1):14–23, 2009.

[6] E. M. Roth, N. Malsch, and J. Multer. Understanding how train dispatchers manage and control trains: results of a cognitive task analysis. Technical report, United States Federal Railroad Administration, 2001. URL https://rosap.ntl.bts.gov/view/dot/8741/dot_8741_DS1.pdf.

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

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

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

[10] H. Khadilkar. A scalable reinforcement learning algorithm for scheduling railway lines. *IEEE Transactions on Intelligent Transportation Systems*, 20(2):727–736, Feb 2019.

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

[12] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016.

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

[14] P. Pellegrini, G. Marlière, and J. Rodriguez. Real time railway traffic management modeling track-circuits. In *Proc. ATOMOS 2012*, pages 23–34. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

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

[16] M. Samà, A. D'Ariano, F. Corman, and D. Pacciarelli. A variable neighbourhood search for fast train scheduling and routing during disturbed railway traffic situations. *Computers & Operations Research*, 78:480–499, 2017.

[17] S. Sinha, S. Salsingikar, and S. SenGupta. An iterative bi-level hierarchical approach for train scheduling. *Journal of Rail Transport Planning & Management*, 6(3):183–199, 2016.

[18] H. Khadilkar. Scheduling of vehicle movement in resource-constrained transportation networks using a capacity-aware heuristic. In *Proc. American Control Conference 2017*, pages 5617–5622. IEEE, 2017.

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

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

[21] 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*, 27:1–15, 2013.

[22] Y. Huang, L. Yang, T. Tang, F. Cao, and Z. Gao. Saving energy and improving service quality: Bicriteria train scheduling in urban rail transit systems. *IEEE Transactions on Intelligent Transportation Systems*, 17(12):3364–3379, 2016.

[23] S. Mackenzie. *Train scheduling on long haul railway corridors*. PhD thesis, University of South Australia, 2010.

[24] Hassan Smaoui, Lahcen Zouhri, Sami Kaidi, and Erick Carlier. Combination of FEM and CMA-ES algorithm for transmissivity identification in aquifer systems. *Hydrological Processes*, 32(2):264–277, 2018.

[25] Patrick MacAlpine, Samuel Barrett, Daniel Urieli, Victor Vu, and Peter Stone. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Proc. AAAI 2012*. AAAI Press, 2012.