A Scalable Reinforcement Learning Algorithm for Scheduling Railway Lines

Harshad Khadilkar

***Abstract*— This paper describes an algorithm for scheduling bidirectional railway lines (both single- and multi-track) using a reinforcement learning (RL) approach. The goal is to define the track allocations and arrival/departure times for all trains on the line, given their initial positions, priority, halt times, and traversal times, while minimizing the total priority-weighted delay. The primary advantage of the proposed algorithm compared to exact approaches is its scalability, and compared to heuristic approaches is its solution quality. Efficient scaling is ensured by decoupling the size of the state-action space from the size of the problem instance. Improved solution quality is obtained because of the inherent adaptability of reinforcement learning to specific problem instances. An additional advantage is that the learning from one instance can be transferred with minimal re-learning to another instance with different infrastructure resources and traffic mix. It is shown that the solution quality of the RL algorithm exceeds that of two prior heuristic-based approaches while having comparable computation times. Two lines from the Indian rail network are used for demonstrating the applicability of the proposed algorithm in the real world.**

***Index Terms*— Machine learning, rail transportation.**

I. Introduction

SCHEDULING problems are an extensively studied class of problems in computer science and operations research. Several versions of scheduling problems have been shown to be NP complete, including JSSP, or the job shop scheduling problem [1]. JSSP aims to define a time-indexed mapping of jobs to resources, while adhering to resource capacity and processing time constraints. The railway problem has been shown in literature to be a 'blocking' version of JSSP [2], [3], where the job (train) must wait in the current resource (track) until the next resource is freed (there is no buffer for storing jobs between two resources). This version of the JSSP is also NP complete [4], with the result that exact solutions require an exponential amount of time for computation. Exact mathematical programming formulations of the railway scheduling problem are available [5], but are not practicable for problems of a realistic scale. An alternative to exact approaches is the use of heuristics [6], where a difficult balance between development effort, computational performance, and solution quality must be struck. This paper describes a third approach, which is closer to optimal than heuristic solutions while still retaining their low computation times (hence greater scalability). It belongs to a class of reinforcement learning algorithms known as table-based Q-learning [7].

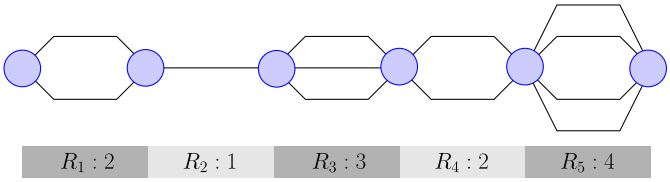

Fig. 1. Illustration of a railway line with stations (R_1, R_3, R_5) and inter-station track sections (R_2, R_4). The number of parallel tracks (denoted by solid lines) available at each station and section is arbitrary. Stations and sections are both treated as generic resources with capacity equal to number of tracks. The circles are all-way connections between neighbouring resources.

Fig. 2. Schematic of the Mumbai suburban rail network, represented as a set of lines (different colours) connected at junctions (from en.wikipedia.org).

A. Problem Context

This paper deals with linear railway networks with multiple parallel tracks, of the type illustrated in Fig. 1. This restriction on topology is reasonable because rail networks are designed in the form of multi-station linear arcs connected at junction stations. Fig. 2 demonstrates this concept using the suburban rail map of Mumbai. The following track usage rules are assumed to apply in this paper. As will be apparent from the algorithm description in subsequent sections, these assumptions are not fundamental to the reinforcement learning procedure; instead, they simplify the description of the procedure.

Manuscript received August 18, 2017; revised January 4, 2018 and February 26, 2018; accepted April 18, 2018. The Associate Editor for this paper was R. Goverde.

The author is with the TCS Research, Tata Consultancy Services, Mumbai 400093, India (e-mail: harshad.khadilkar@tcs.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2018.2829165

- All tracks are bidirectional (can be utilised by trains moving either left to right or vice versa, as per Fig. 1).
- Each track (whether at a station or between two stations) can hold at most one train at a time. Parallel tracks within the same resource may hold one train each, in either direction (known in railway context as *block signalling*) [8].
- Trains can move from any track in one resource to any track in the next resource in their direction of movement (full connectivity). Another train may occupy the freed track after a fixed minimum time (safety margin) has elapsed. This margin allows trains to block two tracks simultaneously, while they are moving from one resource to the other.

B. Instantiating a Scheduling Problem

A specific problem instance begins by defining the resources on the railway line, as given by the number of stations, their order, and the number of parallel tracks (both at each station and between two neighbouring stations). This level of detail is referred to in literature as *mesoscopic* [9]. Besides resource-level information, train movements over the scheduling horizon must be described in one of two ways. The first option is to define a reference timetable which gives the desired arrival and departure time of each train at each station. The second option is to provide the earliest movement times from their current locations (or origin stations), followed by the minimum running times (on track sections between stations) and halt times (at stations) up to the destinations. Note that the running and halt times can be completely heterogeneous: each train may have a different running/halt time in each resource, depending on the length of track, the type of halt, and the type of locomotive. The timetable can be derived by adding the running and halt times of each train to the current time.

Such a timetable may be infeasible if the desired arrival and departure times violate the track usage rules defined earlier. The task of the scheduling algorithm is to adjust the arrival and departure times such that all rules are satisfied, while minimizing an objective called priority-weighted delay (formalised in Sec. III-C). In order to ensure clear differentiation from the desired *timetable*, the feasible train movement plan is referred to in this paper as the *schedule*. This schedule is to be computed for all trains up to their destinations. The basic scheduling problem assumes that the initial system state is empty, and trains are to be introduced at their origin stations at the appropriate times. During operation, a formerly feasible timetable or schedule may become infeasible because of stochastic events such as train delays. This scenario triggers an activity known as rescheduling [10], [11], where a new schedule must be computed based on the current location of all trains. The algorithm described in this paper is tested on both types of situations (scheduling/rescheduling) in Sec. IV.

C. Brief Description of the Proposed Algorithm

Reinforcement learning (RL) [7] works by learning to map the *state* of the system and the choice of available *actions* to long-term *reward* (or penalty). The system in the present context is a set of resources (known as block sections in railway literature) connected to form a railway line. For simplicity, it is assumed that there is a single block section between two stations, so that successive resources alternate between stations and track sections. Each resource has a fixed number of parallel tracks (capacity). The actions correspond to binary decisions (stop/go) for each train when it has completed the required residence time in a given resource. The decisions are based on the state of resources in its local neighbourhood, its priority, and the learned mapping from states and actions to rewards (in the form of *Q-Values*). In this work, the Q-Values intuitively represent the probability of a successful episode termination, which happens when all trains reach their destinations. The termination condition ensures that the resulting solution (schedule) is globally feasible.

D. Contribution of This Study

The methodology described in this paper appears to be the first RL approach to successfully scale to large, realistic, single- and multi-track instances of railway scheduling, based on the literature review in Sec. II. The size of the state-action space is invariant with the size of the problem instance, as described in Sec. III. The update procedure and decision-making logic is easily explainable, unlike black-box algorithms such as deep learning [12]. It has been tested on instances reflecting actual lines in the Indian Railway network, spanning up to 700 km, 60 stations (approximately 120 resources after including track sections), and 440 trains, as described in Sec. IV. Finally, the learning obtained by training on one problem instance is shown to be transferable to other, unrelated instances without extensive retraining. Unlike evolutionary algorithms [11], problem instances can be solved with computation times comparable to deterministic heuristics.

II. Literature Review

The problem of railway scheduling has been extensively studied in literature, both because of the economic significance of delays, and because of its mathematical complexity. Traditional approaches can be divided into three classes. First, some studies use analytical techniques to model broad characteristics such as congestion and delay propagation [13], [14]. However, such studies make simplifying assumptions about the random processes [13], or about regularity/periodicity of the timetables [14]. Such studies provide intuition about the problem, but are not useful for solving practical scheduling instances. Second, there exist formal optimisation methods that are solved using mixed-integer linear [5], [10], [15] or quadratic [16] programming. Exact solutions to such formulations are severely limited in terms of scale. Successful instances are typically limited to a small number of arrival/departure events [16], conflicts [5], or spatial span [15]. Finally, there is a body of literature on heuristics for railway scheduling spanning a period of 20 years [2], [17]–[20]. The most popular out of these is the travel advance heuristic (TAH), which is used for benchmarking the results in Sec. IV. Problems that are common to all heuristic approaches include the suboptimality of generic heuristics, and the high degree of effort required for defining higher-quality specialised solutions.

Machine learning represents an opportunity to overcome the limitations of exact and heuristic methods, but has not yet been widely exploited for railway scheduling. Supervised learning has been used for mimicking human controllers [21], but is limited to conflict resolution. There are studies that use RL or dynamic programming to compute energy efficient speed profiles for single trains [22], [23]. Other learning approaches in the railway domain [24], [25] tend to focus on train marshalling within yards. Recent surveys of algorithms for railway scheduling [10], [26] cover exact approaches, simulation models, constraint propagation, alternative graphs, heuristics, and expert systems, but find no previous RL methods. Instead, one has to rely on solutions for the closely related job-shop scheduling problem [27], [28], but these are few in number, and use learning as a tool for repairing infeasible schedules rather than for scheduling from scratch.

A recent study [29] appears to be the first to use reinforcement learning for railway scheduling. It uses a Q-learning approach similar to the present study, but there are important distinctions between the two approaches. The size of the state space in the prior study can be very large, with possible difficulties in training and in memory management. The specificity of the model to a given infrastructure makes it unclear whether the learning from one instance could be transferred to other problem instances. Furthermore, the study is focussed on recovery from initial delays. By contrast, the proposed approach is focussed on developing schedules from any initial state (with or without delays), is compared against two sophisticated railway-specific heuristics [19], [20], is easy to scale up, and is inherently capable of transfer learning.

Several studies have used reinforcement learning in broadly related areas such as playing card or board games [7], [30], [31] and autonomous driving [32]. However, there are important differences between these contexts and railway scheduling, which make direct application of their solution techniques infeasible. First, unlike board games, the 'moves' in railway scheduling can happen simultaneously for multiple trains, without explicit coordination. Second, the timing of a move is as important as the move itself. Third, unlike autonomous driving, the success or failure of railway scheduling is measured by the collective fate of all the trains involved, so train delays cannot be greedily minimised. Fourth, railway scheduling algorithms frequently run the risk of deadlock [20], [29], where no forward moves are possible for some trains.

Using the above motivation, this paper proposes a reinforcement learning method for railway scheduling. The method is related in principle to a prior study that used reinforcement learning to reach a robot to ride a bicycle [33], in that the action space is reduced and rewards are provided for reaching a goal at the end of the episode. However, the problem being addressed is considerably larger in scale, and needs to be usable in both scheduling as well as rescheduling scenarios. The next section describes the methodology in detail.

III. Methodology

The principal goal of the proposed algorithm is to compute schedules for railway lines (either from scratch or from a given starting state), such that their quality exceeds the output of heuristic approaches, while having comparable online computation requirements. Three challenges need to be overcome to in order to achieve this objective: (i) the algorithm must be able to handle different infrastructure and train service instances, (ii) it must scale to large, realistic railway lines, and (iii) it must manage simultaneously moving trains. In this study, the first challenge is addressed by defining a map from the specific state of the instance to a generalised state space of fixed size. The second challenge is handled by decentralising the decisions for individual trains, and limiting the feature vector to a fixed local horizon around each train. Finally, the ordering of train moves is handled by a discrete event simulator which picks the order using a previously defined deadlock-avoidance heuristic [20]. Each component is described below, and the overall algorithm is given in Sec. III-D.

A. Generalised State Representation

A state vector is computed for each train every time a decision about its next move is to be computed. Relative to the direction of motion, we define resources as being behind (in the direction opposite to the direction of motion) or in front (in the direction of motion) of the train. A user-defined finite number of resources ℓ_b behind each train and ℓ_f in front of each train are used for defining the state vector. These are referred to as *local resources*. Including a few resources *behind* the train in the state definition ensures that overtaking opportunities for fast-moving trains are not missed. The total number of local resources is $(\ell_b + 1 + \ell_f)$. Fig. 3 shows an illustration with $\ell_b = 2$ and $\ell_f = 6$.

The entry in the state vector corresponding to each local resource takes one of R integer values $\{0, 1, 2, \ldots, R-1\}$, referred to as the status S_r of resource r. In this study, $R = 3$ is used for all the test cases, so that $S_r \in \{0, 1, 2\}$. Higher values indicate higher congestion within the resource, and are driven by the number of occupied tracks. Let us define the number of tracks in resource r to be equal to N_r, out of which $T_{r,c}$ tracks contain trains converging with (heading towards) the current train, while $T_{r,d}$ tracks contain trains diverging from (heading away from) the current train. Since at most one train can occupy a given track, we note that $T_{r,c} + T_{r,d} \leq N_r$. The mapping from track occupancy to resource status is,

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

Here, $0 \leq w_c, w_d \leq 1$ are weights that can de-emphasise the effect of converging and diverging trains on the perceived status of a resource. Note that the status of a resource is dependent on where the current train is located, relative to the resource. Intuition behind relation (1) can be obtained by setting $w_c = w_d = 1$ and noting that $R = 3$. In that case, S_r is equal to 0 if at least two tracks in r are unoccupied, 1 if only one track is unoccupied, and 2 if the resource is

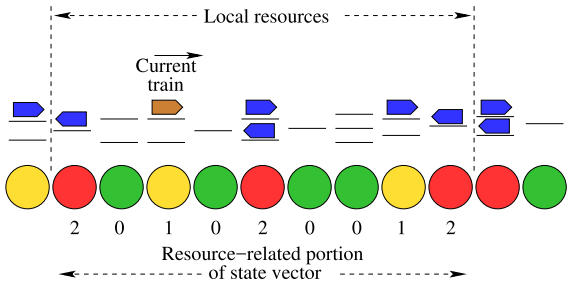

Fig. 3. Mapping train location and direction of movement to resource status, relative to the 'current train'. The train is moving from left to right, as indicated by the tapering portion of the icon. The status of each resource takes one of three values, from 0 (indicating low occupancy) to 2 (indicating high occupancy). Only local resources (in this case, within $\ell_b = 2$ behind and $\ell_f = 6$ in front) are used for building the state vector for each train.

completely full. The flexibility of changing w_c and w_d is used to encourage overtaking of trains in the same direction, or the passing of trains in opposite directions. For example, $w_c < 1$, $w_d = 1$ implies that converging trains contribute less than diverging ones to the perceived congestion, leading the algorithm to pass trains heading in opposite directions more easily than ones heading in the same direction.

In addition to the resource-related entries, the state includes an entry for the priority of the current train. Train priorities are assumed to be static, externally defined integer values, and the contribution of train delays to the objective function is inversely proportional to the priority levels. The relationship is formalised in Sec. III-C. The complete state representation used in this study is a vector $\bar{x}$ of length $(\ell_b+\ell_f+2)$, including the integer priority value and $(\ell_b+1+\ell_f)$ entries for the status of local resources. If we assume that the model accommodates up to P priority levels, the size of the state space is equal to $(P \cdot R^{\ell_b+1+\ell_f})$. Note that this value does not depend on the scale of the problem instance, in terms of the number of trains, the lengths of their journeys, and the number of resources. Also, the state does not include the delay being carried by a train. This value may be included in the state, should it be desired to give dynamically higher priority to delayed trains. However, this aspect is not tested in the current work.

B. Action and Policy Definition

The reinforcement learning procedure maps each state vector to a probability of choosing the *action* to be taken. In this study, the choice of actions in any given state is binary, with 0 representing a decision to *move* the current train to the next resource on its journey, and 1 representing a decision to *halt* in the current resource for a predefined time period (1 minute in this paper). If the train is halted, the decision-making procedure is repeated after the time period elapses. The order in which trains are selected for move/halt decision-making will be discussed in Sec. III-D. At the moment, let us assume that a particular train occupying one track of some resource r has been selected, the state vector has been computed, and the action (move or halt) is to be chosen. In addition to the state vector, the choice of action is driven by the *policy*.

The algorithm uses a slightly modified version of the ϵ-greedy policy [7] for action selection. Given a state, the two

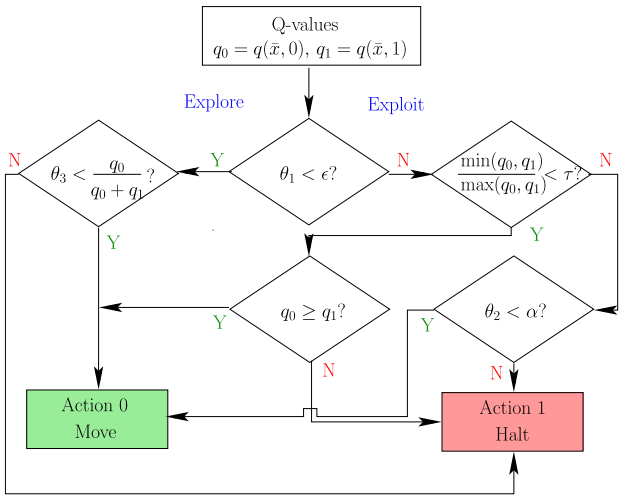

Fig. 4. Flowchart representation of the policy for choosing actions, where θ_i are uniform random variables over the range $[0, 1]$. The value of ϵ remains constant within an episode, and decreases over successive episodes. The left half of the figure is frequently visited when ϵ is close to 1, and corresponds to the exploration phase. The right half of the figure is visited more frequently as ϵ reduces, leading to greater exploitation of previous learning.

possible actions $a \in \{0 : \text{move}, 1 : \text{halt}\}$ result in two unique state-action pairs. Each state-action pair $(\bar{x}, a)$ is associated with a Q-Value $q(\bar{x}, a)$ which quantifies its desirability as described in Sec. III-C. The higher the Q-Value, the higher the desirability of the relevant pair. The ϵ-greedy policy chooses the greedy option (higher Q-Value) with probability $(1 - \epsilon)$, and a randomised action with probability ϵ. The greedy choice corresponds to exploitation of the learning so far, while the randomised choice corresponds to exploration of the state-action space. In exploration mode, the action is chosen with the toss of a biased coin, based on the relative Q-Values q_0 and q_1 of the two actions (see Fig. 4). In the exploit mode, if $q_0 \approx q_1$ within a user-defined threshold τ, a further biased coin toss is used to compute the action. The bias in this case is given by a user-defined aggression parameter α, which controls the probability of choosing 'move' when $q_0 \approx q_1$. If q_0 and q_1 are clearly separated, the action with the higher Q-Value is chosen. The value of ϵ starts at 1 in the first training episode and decreases as more episodes are completed. This moves the policy gradually from exploration towards exploitation.

C. Objective Function, Rewards, and Q-Values

A number of objective functions have been used in the railway scheduling context, in order to achieve goals such as delay reduction, passenger convenience, and timetable robustness [10]. One of the commonly used measures of schedule quality is priority-weighted delay [21]. A delay is defined to be the non-negative difference between the time of an event as computed by the algorithm, and the desired time as specified by the timetable. The priority-weighted average delay is the mean over all trains and all stations of individual delays divided by train priorities. This quantity is used as the objective function (2), but the algorithm can accommodate other measures equally easily (for example, a non-linear function of delays in order to increase fairness

of delay distribution).

$$J = \frac{1}{N_{r,t}} \sum_{r,t} \frac{\delta_{r,t}}{p_t}, \tag{2}$$

where $\delta_{r,t}$ is the delay for train t on departure from resource r, p_t is the priority of train t, and $N_{r,t}$ is the total number of departures in the schedule. Note that this expression includes all events for all trains, for their entire journey.

A typical reinforcement learning construct would use the terms within the summation in (2) as the (negative) reward after completing each move, in order to drive the algorithm towards minimizing J. However, this assignment presents three difficulties in the current context. First, the overall objective J is the sum over all train delays, and a delay assigned to one train affects the delays assigned to other trains. The apportionment of rewards to individual moves is not clear. Second, the magnitude of delays (and hence the theoretical optimum value of J) is different from one problem instance to another. Quantifying rewards directly in terms of delays $\delta_{r,t}$ would create obstacles when transferring the learning from one instance to another. Third, the back-propagation of rewards after the end of the episode is not possible, because the episode can be very long (leading to memory issues).

Instead, this study uses a reward system that is inspired by the *satisficing* philosophy of goal programming [34], [35]. The algorithm maintains a threshold of J as the goal to be achieved in each episode. This maximum acceptable level is set to a proportion $(1+\rho)$ of the minimum J observed thus far, where $\rho > 0$ is a user-defined constant. The threshold becomes tighter as the best known J is improved upon during learning. A reward of $+1$ (success) is given if the sum of the priority-weighted delay is under the current threshold, and -1 (failure) either if it is over the threshold, or if the episode enters deadlock and does not terminate.

The notion of Q-Values is meant to quantify the desirability of state-action pairs, and to discriminate between the range of feasible actions available in any state. In order to retain this property while generalising the learning from any given problem instance, the Q-Values in the current work are defined using the *probability of success* when an episode passes through a given state-action pair. Instead of tracking the entire sequence of state-action pairs in a given episode, a binary indicator variable $b(\bar{x}, a)$ corresponding to each pair $(\bar{x}, a)$ is set to TRUE whenever it is observed in a given episode. Upon termination of the episode, the number of successes (or failures) of all $(\bar{x}, a)$ where $b(\bar{x}, a) = \text{TRUE}$ are incremented[1] by 1. The success probability $\sigma(\bar{x}, a)$ is computed by dividing the number of $+1$ rewards associated with the pair, by the total number of episodes that passed through this pair. If $\mathcal{E}_{\bar{x},a}$ is the number of all episodes that passed through $(\bar{x}, a)$ at least once, and $\mathcal{E}^*_{\bar{x},a}$ is the number of these that ended in success,

$$0 \leq \sigma(\bar{x}, a) = \frac{\mathcal{E}^*_{\bar{x},a}}{\mathcal{E}_{\bar{x},a}} \leq 1. \tag{3}$$

[1]Note: It would have been possible to weight the number of successes by the frequency of observation of a state-action pair. However, some pairs are much more frequently observed than others, and the large variability in learning rates led to unstable results during testing.

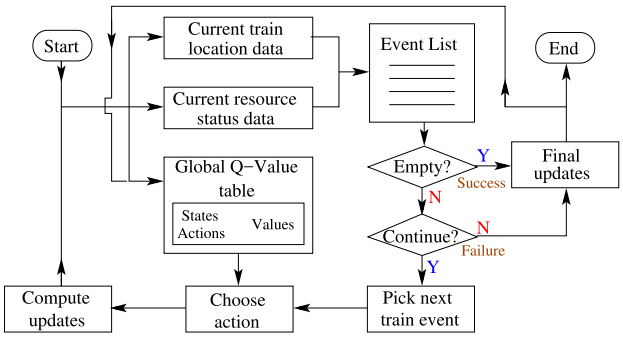

Fig. 5. Flowchart of the algorithm for each scheduling episode.

While $\sigma(\bar{x}, a)$ provides a way to quantify the desirability of a given state-action pair, it does not encapsulate the state trajectory. On the other hand, a core tenet of reinforcement learning is the back-propagation of rewards through the trajectory of state-action pairs (usually upon episode termination). However, in the current context, episodes can be very long, reward is generated only upon episode termination, and state-action pairs for multiple trains are generated simultaneously. Therefore, $\sigma(\bar{x}, a)$ is used in this study as a proxy reward in two ways. First, $\mathcal{E}^*_{\bar{x},a}$ quantifies terminal reward. Second, a one-step back-propagation of $\sigma(\bar{x}, a)$ is carried out while the episode is running. The most recent state-action pair $(\bar{x}, a)$ is retained in memory by *each train*, and the next state-action pair observed by the same train is marked as a 'neighbour' $(\bar{x}', a')$. Every pair $(\bar{x}, a)$ can have several (say M) neighbours $(\bar{x}', a')$ during an episode. The average success rate $\sigma(\bar{x}', a')$ of all M neighbours forms a part of the Q-Value $q(\bar{x}, a)$:

$$q(\bar{x}, a) = w\sigma(\bar{x}, a) + (1-w) \sum_{m=1}^{M} \frac{\sigma(\bar{x}'_m, a'_m)}{M},$$

where w is a weighting factor between the success rate of a given pair, and the average success rate of its M neighbours. These M neighbours are not necessarily unique pairs, and thus neighbouring pairs that are more frequently observed have a greater contribution to the average value. Updating the average is memory-efficient, since it is only required to store the current values of the average and of M.

Intuitively, the definition of $q(\bar{x}, a)$ represents the probability of pair $(\bar{x}, a)$ being involved in a successful episode. The two components of $q(\bar{x}, a)$ can be initialised with some prior estimates, and are updated at different rates during reinforcement learning. The first portion $\sigma(\bar{x}, a)$ is incremented or decremented using (3) for all observed pairs at the end of an episode. The second component (success rate of neighbours) is updated for one pair after each decision within an episode.

D. Description of the Integrated Algorithm

The integrated reinforcement learning algorithm is driven by a discrete event simulator, as illustrated in Fig. 5. The simulator runs through several episodes during training. At the beginning of every episode, the initial locations of all the trains are reset to their original values. It is assumed that trains that

have not yet started, or have finished their journeys, do not occupy any of the tracks. Following the train-to-resource mapping, the simulator creates a list of events for processing, one corresponding to each train (whether already running or yet to start its journey). Each event in the list contains the following information: the *time* at which to process the event, the *train* to which it corresponds, the *resource* where the train is currently located, the *last observed state-action pair* for the train (empty if the train is yet to start), and the *direction* of the train journey. If a train is standing at a station, the event processing time corresponds to the earliest time at which the train can depart, as defined by its minimum halt time at the station and by any departure time constraints enforced for passenger convenience. If it is running between two stations, the event processing time corresponds to the earliest time at which it can arrive at the next station, as defined by the length of the track and the train running speed. If the train is yet to start, the event processing time is the time at which it is expected at the starting station.

At each step, the algorithm moves the simulation clock to the earliest time stamp in the event list. If multiple events are to be processed at the same time stamp, they are handled sequentially. This sequence is decided by a deadlock-avoidance heuristic from prior literature [20]. Intuitively, the sequence of events is ordered in such a way as to process trains in the most congested resources first. The lower the number of free tracks in a resource, the higher the congestion, and the earlier the processing of a train occupying that resource. Once an event is selected for processing, the state vector is constructed for the relevant train. An action is chosen using the state vector, the corresponding Q-values, and the randomised procedure for deciding whether to explore or to exploit the learning so far.

The action is implemented by the simulation engine, which updates (i) the data structures (train location, resource occupancy), (ii) the event list (including the updated train position and time when the next decision is due), and (iii) the Q-value table. The simulation engine ensures that all decisions are feasible (at least one track free in the next resource). Infeasible decisions are penalised by setting the success probability of the state-action pair to 0, and reversing the decision. Note that only a 'move' decision can be invalid, since a train can always choose to halt in its current resource (no pre-emption). The algorithm proceeds either until all trains successfully reach their destinations, or until a failure condition is satisfied.

The episode is called a success if all trains finish their journeys with an objective as per (2) that is within $(1+\rho)$ of the best value so far. The episode is called a failure if J is above this threshold, or if it fails to terminate due to a deadlock between trains. A deadlock is identified if all the tracks within two neighbouring resources are occupied by trains heading towards each other (head-on situation with no passing room). A number of scheduling episodes are simulated, in order to let the Q-Values stabilise to their final values. More details about this process are provided in Sec. IV.

IV. Results

The methodology as described in Sec. III is tested on five instances of the railway scheduling problem, of varying scale. Sec. IV-A illustrates the procedure using a toy instance (HYP-1). Sec. IV-B and IV-C describe two larger hypothetical instances (HYP-2 and HYP-3), and two realistic instances based on portions of the Indian Railway network (the Konkan and Ajmer railway lines).

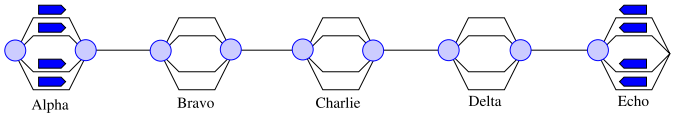

Fig. 6. Initial state of the HYP-1 problem instance.

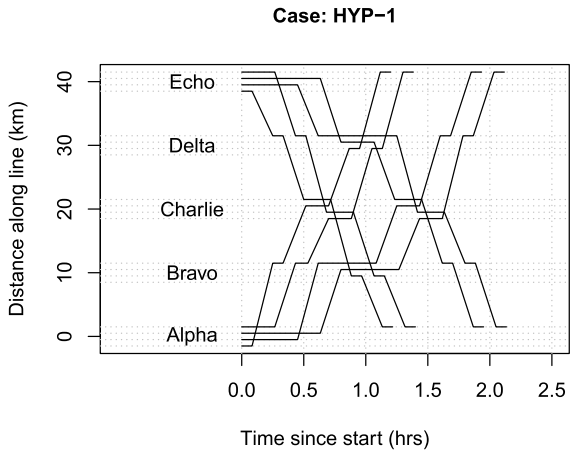

Fig. 7. Schedule computed using reinforcement learning, for a small test case with 8 trains and 5 stations. Each solid line marks the trajectory of one train.

All results use a look-forward of $\ell_f = 6$ resources, and a look-back of $\ell_b = 2$ resources. The status of each resource takes one of three values: 0, 1, or 2 ($R = 3$). The weight on converging trains is $w_c = 0.9$, while that on diverging trains is $w_d = 1$. The maximum number of priority levels is $P = 3$. The threshold for checking whether $q_0 \approx q_1$ (as per Sec. III-B) is $\tau = 0.9$, and the aggression parameter is $\alpha = 0.9$. The threshold for determining the maximum acceptable J is $\rho = 0.25$. The algorithm is trained on each problem instance for several hundred episodes with the same initial train locations. Test results are computed using randomly perturbed versions of the training timetables, as explained in Sec. IV-C. All experiments are carried out in R [36] running on an Ubuntu 4-core machine with 4 GB RAM.

A. Illustration Using a Toy Problem Instance

The simple hypothetical instance HYP-1 consists of 8 trains travelling 5 stations each. This instance is analogous to the rescheduling scenario from Sec. I, since all trains are already in the system at the start. Four trains heading left to right are initially located in station Alpha, as shown in Fig. 6, while the others are in station Echo, heading right to left. Each station contains 4 parallel tracks, and there is a single track between stations. Only for HYP-1, the trains all have the same priority, running times between stations, and halt times at stations.

Each training episode computes a schedule of the type shown in Fig. 7. The figure shows time on the x-axis and distance on the y-axis. Each solid line shows the trajectory of

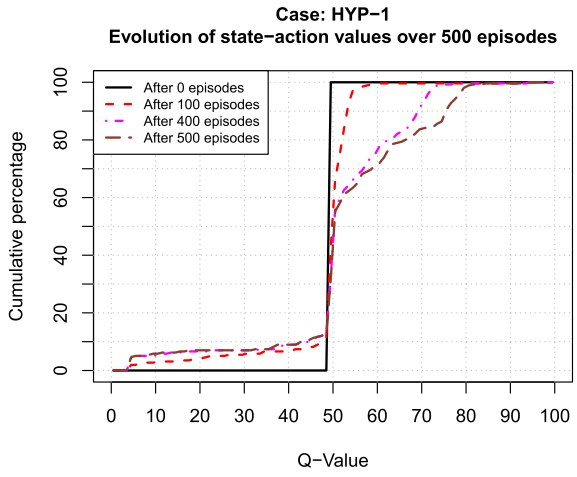

Fig. 8. Evolution of Q-Values for the HYP-1 test case, shown as a cumulative distribution. All Q-Values are initialised to 50% at the start of training.

one train as it moves from one end of the line to the other. The horizontal portions correspond to halts at stations, while the inclined portions denote movement between stations. Since there is a single track between successive stations and only one train can occupy it at a time, inclined lines cannot cross each other in a feasible schedule. The horizontal dotted lines indicate specific tracks within station resources, and no two solid lines are allowed to overlap within these tracks.

Since HYP-1 contains trains of a single priority, the number of states is $P \cdot R^{(\ell_b+1+\ell_f)} = 19683$, with two actions per state. Each state-action pair is initialised with a success rate $\sigma(\bar{x}, a) = 50\%$, leading to an initial Q-Value of 0.5 or 50%. The evolution of the Q-values during training is shown in Fig. 8, plotting only the pairs that were visited at least once. The x-axis shows the Q-Value (in percentage terms) of the states, and the y-axis shows the cumulative percentage of states that fall at or below the corresponding Q-Value. At the beginning, there is a step change at 50. The curve becomes smoother with training as the state-action space is explored. The difference in the curves for 0 and 100 episodes is substantial for all Q-Values. The curves after 400 and 500 episodes are nearly identical for low Q-Values, but noticeably different for higher Q-Values. The training for this instance has been deliberately stopped at 500 episodes, in order to emphasise the relatively slow rate of learning when starting with all Q-Values at 50%. In subsequent discussion, this aspect will be explained further.

B. Testing on Larger Problem Instances

The toy instance from Sec. IV-A shows that the proposed approach is able to learn Q-Values (and hence the policy for choosing actions) at a small scale. However, when instances of a realistic size and complexity are run with the same initial Q-Values (50%), they require a large number of training instances to start producing feasible solutions. This happens because the larger problem instances require several thousand decisions to be made 'correctly' for successful completion. When such decisions are made purely randomly, the instances frequently end with trains in deadlock situations.

TABLE I
TEST CONDITIONS FOR EXPLICIT INITIALISATION OF Q-VALUES

#	States satisfying	Action	Init. Q-V (0-100)
1	Next resource is full ($S_{r,\text{next}} = R - 1$)	Move Stop	0 50
2	At least three consecutive resources are full ($S_r = S_{r+1} = S_{r+2} = R - 1$)	Move Stop	10 15
3	Next resource almost full ($R - 2$), next-but-one is full ($R - 1$)	Move Stop	15 50
4	Average status of upcoming resources is between 0.5 and 1.0 (moderately empty)	Move Stop	85 50
5	Average status of upcoming resources is less than 0.25 (almost empty)	Move Stop	95 50

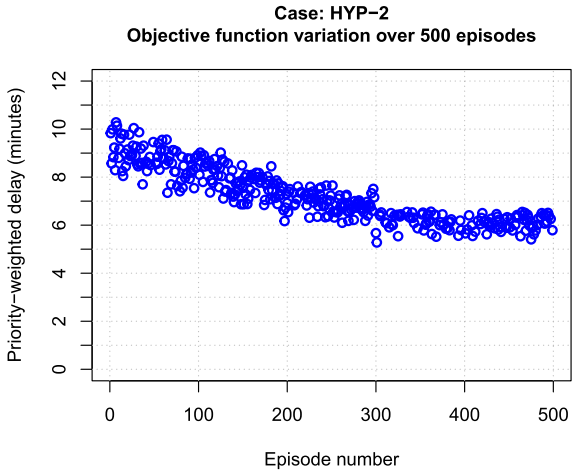

Fig. 9. Scatter plot of objective function for 500 episodes of HYP-2.

A simple work-around to this problem is to encode basic deadlock-avoidance rules in the algorithm, by initialising Q-Values of the relevant state-action pairs to appropriate values. A set of five conditions is evaluated for each state in the precedence order given in Table I, and the Q-Values for the state-action pairs that satisfy at least one of these conditions are initialised as shown. The reasoning behind these conditions is to discourage the *move* action choice when resources in front of a train have higher status values (indicating high upcoming congestion), and to encourage it when the resources in front are relatively free. The Q-Values for the *move* and *stop* actions need not sum to 100. In the explore mode, the action choice is computed randomly in proportion to the respective values.

Initialising the Q-Values leads to a smooth training exercise, as illustrated in Fig. 9. This figure shows a scatter plot of the objective function (y-axis) over the course of 500 training episodes (x-axis) on a moderately-sized problem instance with 60 trains and 11 stations (HYP-2), solved in a clean-slate scheduling situation (in the initial state, all trains are yet to start their journeys). The objective value reduces from 10 minutes in the first few episodes to a nearly constant 6 minutes in the last few episodes. Analogous to Fig. 8, the evolution of Q-Values during training is shown in Fig. 10,

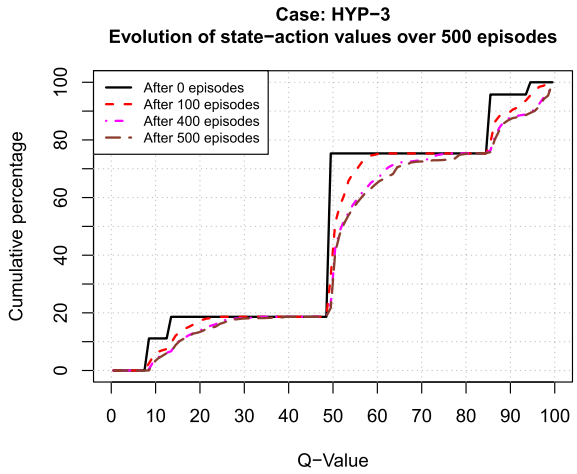

Fig. 10. Evolution of Q-Values for HYP-3, with initialisation.

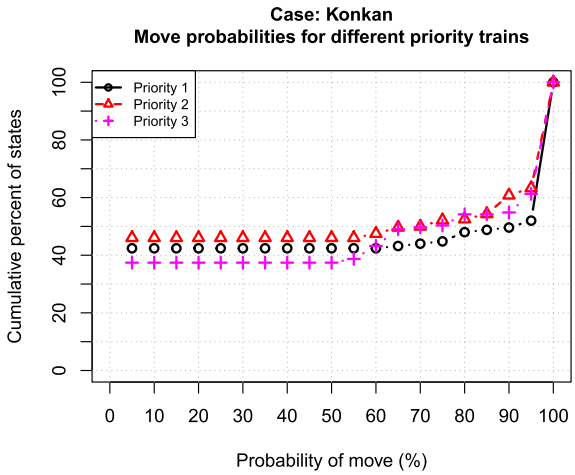

Fig. 11. Move probability distribution for different priority trains in the Konkan test case. Results computed using Q-Values after 500 episodes.

for a problem instance (HYP-3) that uses the same infrastructure as HYP-2, but has twice the number of trains. Note that the solid line in Fig. 10, which shows the distribution of initial Q-Values, is more staggered than the one in Fig. 8. The distribution of Q-Values after 400 and 500 episodes is nearly identical, indicating that the training procedure has nearly stabilised. Some amount of change is still seen in the band of Q-Values between 50% and 80%. The same plot for HYP-2 (not shown for brevity) shows similar characteristics, but has even smaller differences between the curves for 400 and 500 episodes. Overall, a comparison between Fig. 8 and Fig. 10 shows that the Q-Value initialisation contributes to faster training.

The initialised Q-Values were also used to compute schedules for two railway lines within the Indian railway network, data for which was obtained from public sources [37]. The Konkan railway line runs over a distance of 700 km through 59 stations (117 resources, including 58 track sections). The journey times for slower trains on this line are approximately 24 hours. Computing schedules for time horizons much shorter than this value may lead to situations where the short-term solutions are good but lead to deadlocks later in the journey. Therefore, the scheduling horizon was chosen to be 24 hours and included all 85 planned trains, accounting for over 5400 arrival and departure events (not all trains traverse the entire length of the line). The second test instance is a mixed single/double track line near Ajmer in northern India. From the desired timetable, the traffic congestion had peaks and troughs with a periodicity of approximately 24 hours. In order to ensure that the effect of one peak was not affecting the next one, a scheduling horizon of 72 hours was chosen. It included 444 trains and 52 stations (103 resources, including 51 track sections), accounting for over 26000 events in total.

The discrimination between trains of different priority in the Konkan instance is illustrated by the plots shown in Fig. 11. The x-axis shows the probability of choosing the *move* action in a given state (note the difference from the x-axis in previous figures), while the y-axis shows the cumulative percentage of states with a probability at or below the relevant x-value.

TABLE II

OBJECTIVE VALUES FOR 4 TEST CASES. NUMBERS IN PARENTHESES ARE COMPUTATION TIMES IN MINUTES. ALL RESULTS ARE AVERAGED OVER 10 INDEPENDENT TEST TIMETABLES EACH, GENERATED BY RANDOMLY PERTURBING THE ORIGINAL TIMETABLE USED FOR TRAINING. TAH-FP AND TAH-CF ARE HEURISTIC ALGORITHMS FOR COMPARISON

Name	Stns.	Trains (sorted by priority)	Events	RL (min)	TAH-FP (min)	TAH-CF (min)
HYP-2	11	15, 45, 0	1320	4.04 (0.35)	5.37 (0.12)	6.62 (0.12)
HYP-3	11	40, 80, 0	2640	19.00 (1.19)	- (5+)	152.32 (1.31)
Konkan	59	6, 49, 30	5418	4.91 (2.90)	5.28 (0.43)	5.60 (0.45)
Ajmer	52	27,289,128	26258	2.04 (10.58)	17.11 (11.84)	5.55 (3.81)

A substantial fraction of the states have a 0% probability of choosing to move, corresponding to the states where the algorithm has identified infeasible moves. The largest number of these satisfy condition 1 in Table I. Of greater interest are the states with a high ($> 50\%$) probability of move. The curve for priority 3 (least weight in the objective function) shows a jump between 55% and 65%, indicating the presence of a large fraction of states in this region. On the other hand, priority 2 shows a similar jump between 85% and 90%, while priority 1 shows this jump in the 95% to 100% range. Locations where a large number of states are present, indicate that given a similar set of resource statuses, trains with priority 1 have higher move probabilities than trains with priority 2, which in turn have higher move probabilities than trains with priority 3.

C. Performance Comparison With Prior Methods

As explained in Sec. I, exact optimisation methods do not scale to realistic problem instances, and are therefore unavailable for benchmarking purposes. Instead, Table II

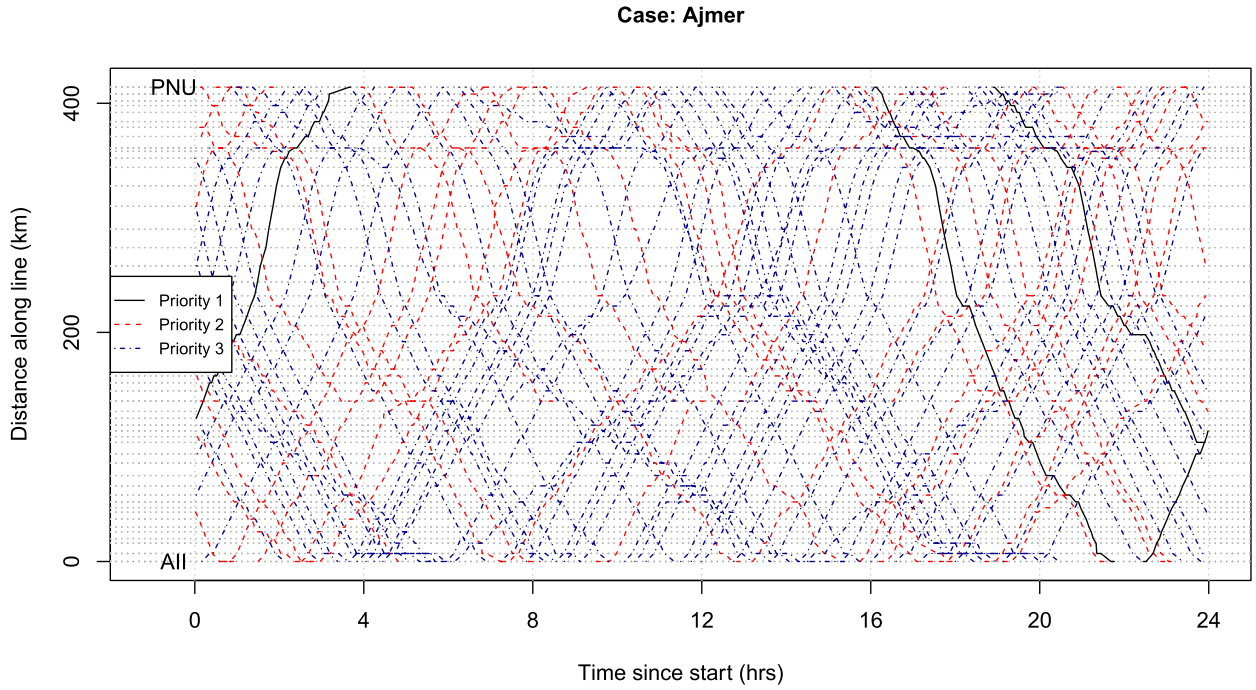

Fig. 12. Snapshot spanning 24 hours of the schedule computed using reinforcement learning, for the Ajmer railway line.

compares the results of the reinforcement learning approach with those produced by two heuristics from prior literature: Fixed Priority Travel Advance Heuristic (TAH-FP) [19] and Critical First Travel Advance Heuristic (TAH-CF) [20]. Of the two, TAH-FP is more greedy, and picks train moves by a strict priority order and a 1-step lookahead. On the other hand, TAH-CF is focussed on deadlock avoidance, trading off schedule quality for a higher probability of successful completion. It uses global resource information for picking train moves. Both heuristics iteratively pick a train and move it one station forward at a time, and 'backtrack' (revert the move) in case of deadlock. The same move is attempted again at a later point of time.

Table II shows average objective values and computation times for all three algorithms, averaged over 10 perturbed versions of the timetable used for training. The perturbations correspond to changes in the desired departure times of trains, drawn from a uniform random distribution over $[-30, 30]$ minutes for each train. The resource layout remains the same as the training instance. The advantages of RL are more pronounced as the test instances become more complex. HYP-2 is a hypothetical instance with 11 stations and 60 trains (15 of priority 1 and 45 of priority 2). In this case, the priority-weighted delay for RL is 24% lower than for TAH-FP, and 39% lower than TAH-CF. However, the computation time for RL is three times that of the heuristics. This gap is seen because HYP-2 has a moderate level of traffic density, with the most-utilised resource occupied 31% of the time. As a result, both TAH-FP and TAH-CF rarely need to backtrack on previous moves, reducing the number of iterative moves.

The trends are different for HYP-3, where the same set of resources are expected to handle twice the number of trains. RL is able to find schedules with a priority-weighted delay of 19 minutes, while requiring 1.19 min for computation. The most-occupied resource is utilised for 57% of the time. On the other hand, both heuristics have severely degraded performance. TAH-CF results in a priority-weighted delay of 152.32 minutes and a computation time larger than RL. TAH-FP is unable to find a feasible solution within 5 minutes. This outcome is a consequence of the heuristics frequently backtracking on previous moves, and repeating the same forward-move computations several times. Since TAH-CF is more careful in its choice of moves, it requires fewer backtracks than TAH-FP. Nevertheless, the computation time is in excess of RL which moves only in the forward direction (no backtracks).

For the Konkan instance, RL results in a priority-weighted delay of 4.9 min averaged over 10 perturbed timetables. The busiest resource is occupied for 43% of the time. Note that the occupancy of the busiest resource falls between the values for HYP-2 and HYP-3. TAH-FP results in an objective value of 5.28 min, while TAH-CF achieves a value of 5.60 min. For the Ajmer instance, the RL approach results in a value of 2.04 min and an average computation time of 10.58 min. The busiest resource is occupied for 41% of the time. TAH-FP is unable to find a feasible schedule within 30 min for six out of the ten perturbed timetables, and the average objective value for the other four instances is 17.11 min. TAH-CF finds feasible schedules in eight out of ten instances, but its objective value is more than twice that of RL. The difference in scale between this test case and the toy example can be visualised by comparing Fig. 12 with Fig. 7. Each horizontal dotted line in Fig. 12 corresponds to a station (different tracks at a station are not shown separately). The region near the 200 km mark

TABLE III
TEST OF TRANSFER LEARNING CAPABILITY. RL-SELF SHOWS RESULTS WHEN THE ALGORITHM IS BOTH TRAINED AND TESTED ON THE SECOND COLUMN. RL-TRANS SHOWS RESULTS WHEN THE ALGORITHM IS TRAINED ON THE FIRST COLUMN, AND IS TESTED ON THE SECOND COLUMN

Trained on	Tested on	RL-self (min)	RL-trans (min)	TAH-FP (min)	TAH-CF (min)
Konkan	Ajmer	2.04	2.13	17.11	5.55
Ajmer	Konkan	4.91	4.97	5.28	5.60
HYP-2	HYP-3	19.00	18.01	-	152.32
HYP-3	HYP-2	4.04	5.02	5.37	6.62
Ajmer	HYP-3	19.00	19.91	-	152.32
Konkan	HYP-3	19.00	19.22	-	152.32

between hours 12-16 is suggestive of peak congestion, even though some of it is regularised by the scheduling algorithm. There is another such region around the 400 km mark and hour 20, further complicated by the presence of a high priority train.

D. Transfer Learning

The generalisation capability of the proposed RL procedure is tested by using the Q-Values trained on one problem instance, and computing schedules for a different instance. Table III shows the results for six such combinations. The RL-self column lists the objective values when the training and testing are carried out on the same instance. RL-trans lists the objective values when the Q-Values from the instance in the first column were used to schedule the instances in the second column. The objective values for RL-self and RL-trans are similar. The small performance gaps are a result of the RL algorithm learning specific quirks of a given instance, when trained and tested on the same instance. However, RL-trans is still competitive in comparison with TAH-FP and TAH-CF. Training on HYP-2 and testing on HYP-3 actually outperforms the version trained on HYP-3, possibly because training on the smaller instance has stabilised to a greater extent.

V. CONCLUSION

This paper described a reinforcement learning approach for scheduling trains on railway lines. The focus of the paper was on developing a methodology that could handle large, realistic problem instances in computation times comparable with heuristic approaches, but with better schedule quality. The results given in the previous section showed that this goal was achieved in at least four test scenarios, including ones based on real timetables. The advantages of using RL rather than heuristics were, (i) RL could directly utilise the objective values for guiding the learning procedure, (ii) it could customise the Q-Values to take advantage of the characteristics of specific problem instances, (iii) it could generalise from one problem instance to another, and (iv) it did not require a great deal of domain expertise for defining the scheduling rules.

In the future, the goal is to test this approach with more parameter variations (such as different values of ℓ_f and ℓ_b), and to extend it to handle networks with connecting lines. Additionally, it may be possible to utilise the proposed algorithm for generic scheduling and planning problems, many of which have similar underlying structure as railway scheduling. In instances where even the generalised state space is too large to model using lookup tables, one may choose to use function approximation methods such as deep Q-learning for modelling.

REFERENCES

[1] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.

[2] A. DâĽ™Ariano, D. Pacciarelli, and M. Pranzo, "A branch and bound algorithm for scheduling trains in a railway network," *Eur. J. Oper. Res.*, vol. 183, no. 2, pp. 643–657, 2007.

[3] S. Liu and E. Kozan, "Scheduling trains as a blocking parallel-machine job shop scheduling problem," *Comput. Oper. Res.*, vol. 36, no. 10, pp. 2840–2852, 2009.

[4] X. Cai and C. J. Goh, "A fast heuristic for the train scheduling problem," *Comput. Oper. Res.*, vol. 21, no. 5, pp. 499–510, 1994.

[5] A. Higgins, E. Kozan, and L. Ferreira, "Optimal scheduling of trains on a single line track," *Transp. Res. B, Methodol.*, vol. 30, no. 2, pp. 147–161, 1996.

[6] A. Higgins, E. Kozan, and L. Ferreira, "Heuristic techniques for single line train scheduling," *J. Heuristics*, vol. 3, no. 1, pp. 43–62, 1997.

[7] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2012.

[8] R. J. Hill, "Electric railway traction. IV. Signalling and interlockings," *Power Eng. J.*, vol. 9, no. 4, pp. 201–206, Aug. 1995.

[9] S. de Fabris, G. Longo, G. Medeossi, and R. Pesenti, "Automatic generation of railway timetables based on a mesoscopic infrastructure model," *J. Rail Transp. Planning Manage.*, vol. 4, nos. 1–2, pp. 2–13, 2014.

[10] W. Fang, S. Yang, and X. Yao, "A survey on problem models and solution approaches to rescheduling in railway networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 6, pp. 2997–3016, Dec. 2015.

[11] L. Chen, C. Roberts, F. Schmid, and E. Stewart, "Modeling and solving real-time train rescheduling problems in railway bottleneck sections," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 4, pp. 1896–1904, Aug. 2015.

[12] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[13] E. R. Petersen, "Over-the-road transit time for a single track railway," *Transp. Sci.*, vol. 8, no. 1, pp. 65–74, 1974.

[14] O. Frank, "Two-way traffic on a single line of railway," *Oper. Res.*, vol. 14, no. 5, pp. 801–811, 1966.

[15] P. Pellegrini, G. Marlière, and J. Rodriguez, "Optimal train routing and scheduling for managing traffic perturbations in complex junctions," *Transp. Res. B, Methodol.*, vol. 59, pp. 58–80, Jan. 2014.

[16] H. Niu, X. Zhou, and R. Gao, "Train scheduling for minimizing passenger waiting time with time-dependent demand and skip-stop patterns: Nonlinear integer programming models with linear constraints," *Transp. Res. B, Methodol.*, vol. 76, pp. 117–135, Jun. 2015.

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

[18] J. Medanic and M. J. Dorfman, "Efficient scheduling of traffic on a railway line," *J. Optim. Theory Appl.*, vol. 115, no. 3, pp. 587–602, 2002.

[19] S. K. Sinha, S. Salsingikar, and S. SenGupta, "An iterative bi-level hierarchical approach for train scheduling," *J. Rail Transp. Planning Manage.*, vol. 6, pp. 183–199, Dec. 2016.

[20] H. Khadilkar, "Scheduling of vehicle movement in resource-constrained transportation networks using a capacity-aware heuristic," in *Proc. Amer. Control Conf.*, Seattle, WA, USA, May 2017, pp. 5617–5622.

[21] S. Dündar and I. Şahin, "Train re-scheduling with genetic algorithms and artificial neural networks for single-track railways," *Transp. Res. C, Emerg. Technol.*, vol. 27, pp. 1–15, Feb. 2013.

[22] S. Lu, S. Hillmansen, T. K. Ho, and C. Roberts, "Single-train trajectory optimization," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 2, pp. 743–750, Jun. 2013.

[23] J. Yin, D. Chen, and L. Li, "Intelligent train operation algorithms for subway by expert system and reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 6, pp. 2561–2571, Dec. 2014.

[24] Y. Hirashima, "A reinforcement learning system for transfer scheduling of freight cars in a train," in *Proc. Int. Multiconf. Eng. Comput. Sci.*, Hong Kong, Mar. 2010, pp. 1–6.
[25] Y. Hirashima, "A reinforcement learning method for train marshaling based on movements of locomotive," *IAENG Int. J. Comput. Sci.*, vol. 38, no. 3, pp. 242–248, 2011.
[26] C. Turner, A. Tiwari, A. Starr, and K. Blacktop, "A review of key planning and scheduling in the rail industry in Europe and UK," *Proc. Inst. Mech. Eng.*, vol. 230, no. 3, pp. 984–998, 2016.
[27] W. Zhang and T. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *Proc. Int. Joint Conf. Artif. Intell.*, Montreal, QC, Canada, 1995, pp. 1–7.
[28] W. Zhang and T. Dietterich, "High-performance job-shop scheduling with a time-delay TD(λ) network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 8. 1996, pp. 1024–1030.
[29] D. Šemrov, R. Marsetic, M. Žura, L. Todorovski, and A. Srdic, "Reinforcement learning approach for train rescheduling on a single-track railway," *Transp. Res. B, Methodol.*, vol. 86, pp. 250–267, Apr. 2016.
[30] A. Grzybowski, "Monte Carlo analysis of risk measures for blackjack type optimal stopping problems," *Eng. Lett.*, vol. 19, pp. 147–154, Aug. 2011.
[31] J. Boyan, "Modular neural networks for learning context-dependent game strategies," M.S. thesis, Comput. Speech Lang. Process, Univ. Cambridge, England, U.K., 1992.
[32] S. Shalev-Shwartz, S. Shammah, and A. Shashua. (2016). "Safe, multi-agent, reinforcement learning for autonomous driving." [Online]. Available: https://arxiv.org/abs/1610.03295
[33] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proc. Int. Conf. Mach. Learn.*, Jul. 1998, pp. 463–471.
[34] A. Charnes and W. W. Cooper, "Goal programming and multiple objective optimizations: Part 1," *Eur. J. Oper. Res.*, vol. 1, no. 1, pp. 39–54, 1977.
[35] M. Tamiza, D. Jones, and C. Romero, "Goal programming for decision making: An overview of the current state-of-the-art," *Eur. J. Oper. Res.*, vol. 111, no. 3, pp. 569–581, 1998.
[36] R Core Team, "R: A language and environment for statistical computing," R Found. Stat. Comput., Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org/
[37] India Rail Info. (Jan. 2015). *Train Timetable and Operational Data Archives*. [Online]. Available: http://indiarailinfo.com/

Harshad Khadilkar received the bachelor's degree in aerospace engineering from IIT Bombay, Bombay, in 2009, and the master's and Ph.D. degrees in aeronautics and astronautics from Massachusetts Institute of Technology, in 2011 and 2013, respectively. He is currently a Scientist with TCS Research, Tata Consultancy Services, Mumbai. His research interests lie in control applications for networked systems, especially in the domains of transportation, energy, and supply chains.