Reinforcement Learning in Railway Timetable Rescheduling

Yongqiu Zhu[1], Hongrui Wang[2] and Rob M.P. Goverde[3]

Abstract— **Real-time railway traffic management is important for the daily operations of railway systems. It predicts and resolves operational conflicts caused by events like excessive passenger boardings/alightings. Traditional optimization methods for this problem are restricted by the size of the problem instances. Therefore, this paper proposes a reinforcement learning-based timetable rescheduling method. Our method *learns* how to reschedule a timetable off-line and then can be applied online to make an optimal dispatching decision immediately by sensing the current state of the railway environment. Experiments show that the rescheduling solution obtained by the proposed reinforcement learning method is affected by the state representation of the railway environment. The proposed method was tested to a part of the Dutch railways considering scenarios with single initial train delays and multiple initial train delays. In both cases, our method found high-quality rescheduling solutions within limited training episodes.**

I. Introduction

Railway operations rely on a timetable that specifies the arrival/departure time of each train at each station satisfying all operational requirements (e.g. safety distances between running trains). In an ideal situation, trains are operated exactly according to the planned timetable to provide travellers with punctual services. In practice, however, there will always be daily variations and disturbances. For example during peak hours, the dwell time of a train at a station could be extended due to excessive passenger alighting and/or boarding. In this circumstance, the train will depart from this station with a certain delay (*primary* delay). If the delay is large enough, it may cause follow-up delays to the arrivals and/or departures of the same train at the downstream stations, and propagate also to following trains via route conflicts or transfer connections (*secondary* delay). Therefore when a delay is detected, it is necessary to predict the potential conflicts and resolve them (if any) with the aim of e.g., minimizing the total secondary delays. This problem is generally called Train Timetable Rescheduling (TTR), which has been extensively studied in the literature. An overview of models and algorithms for TTR is presented in [1].

Most literature formulates TTR as an Alternative Graph (AG) model [2], [3], or a Mixed Integer Linear Programming (MILP) model [4], [5]. These models are often solved by heuristic algorithms in order to obtain feasible solutions in a timely manner. To speed up the computation further, a parallelized algorithmic approach incorporating a heuristic algorithm has been developed in [6]. For the same reason, a concept of ‘dynamic impact zone’ has been proposed in [7] to decompose the TTR problem in space and time so that only a limited number of operational conflicts need to be resolved. Although traditional optimization methods are constantly improved in terms of computation efficiency, they are still restricted by the size of problem instances. To overcome this limitation, a few studies [8]-[12] applied machine learning techniques to realize TTR, in particular the technique of Reinforcement Learning.

Reinforcement learning is learning how to map states of an environment to actions so that the cumulative reward of these actions can be maximized [13]. The learner (generally called *agent*) autonomously explores which actions lead to the most reward via a trial-and-error learning process, which can be seen as a process of training the agent. An agent trained with sufficient samples is able to make an optimal decision given the state of the environment. The basic idea of reinforcement learning makes it promising for real-time problems since it has learned already how to solve the problem beforehand, and the extensive learning experiences ensures its performance in terms of solution quality.

In the literature of reinforcement learning-based TTR [8]-[12], a traffic controller is commonly considered as the agent who take actions by sensing the current state of the railway environment. In [8], the actions are deciding the sequence of trains departing from the stations. The state is represented by the actual departure and arrival times of all trains at all stations. This way of representing a state may limit the applicability of the trained agent to other railway networks, in which different train schedules are used. In [9], the actions are deciding the sequence of trains passing through a junction. The state is represented by the trains that are still waiting for passing through the junction. The focus is on one isolated junction of a double-track railway line. An extension is made in [10] to include multiple junctions, which are considered independently of each other without traffic coordination. In [11], re-ordering and re-timing trains are both considered for a single-track railway line. The actions are controlling the signals along the tracks by turning them to red (stop) or green (proceed) color so that the movements of trains are under control. The state is represented by the availability of all tracks, the current locations of all trains in the system, and the current system time. In [12], single-track and double-track railway lines can both be handled. The actions are controlling train movements by deciding whether a train should wait at the current resource or move forward to the next resource. A

[1]Yongqiu Zhu is with the Department of Transport and Planning, Delft University of Technology, Delft, The Netherlands `y.zhu-5@tudelft.nl`

[2]Honrgui Wang is with the Section of Railway Engineering, Delft University of Technology, Delft, The Netherlands `h.wang-8@tudelft.nl`

[3]Rob M.P. Goverde is with the Department of Transport and Planning, Delft University of Technology, Delft, The Netherlands `r.m.p.goverde@tudelft.nl`

resource is a station or an open track segment (the area between two neighbouring stations). It is assumed that a track can only be occupied by one train at a time, which mostly holds to the tracks within stations but not to the tracks in open track segments (hereinafter referred to as segment for simplicity). The state is represented by the availability of a few resources close to the train currently concerned, which significantly reduces the state space compared to [11]. For a railway network with multiple train lines operating on the same corridors periodically, this state representation is not sufficient for the agent to distinguish between different situations. As a result, the agent may be trained to take the same actions in different situations in which the agent should have taken different actions to obtain a better solution.

This paper applies reinforcement learning to enable TTR inspired by [12]. One of our contributions is considering a more realistic railway operation where multiple trains are allowed to occupy the same open track simultaneously, as long as these trains are running in the same direction and separated safely. Another contribution is that the railway environment is effectively represented, particularly for periodic heterogeneous train services. We will show by numerical experiments that the state representation affects the solution quality. The remainder of this paper is organized as follows. In Section II we give a brief introduction to the reinforcement learning technique adopted in this paper: *Q-learning*. Then, Section III explains how we apply Q-learning to realize TTR. The proposed method is tested in Section IV, and finally Section V concludes the paper and points out future directions.

II. Basic concept of Q-learning

Q-learning is a model-free reinforcement learning method, which does not need a state transition function of the environment [13]. It is thus suitable for a complex environment that needs much effort to be accurately modelled. Due to the complexity of the railway environment, this paper uses Q-learning.

The key elements of Q-learning include a set of states S, a set of actions A per state, a reward function $R(s,a)$, and a state-action value function $Q\left(s,a\right)$. The reward function $R(s,a)$ returns a reward of taking action a at state s. The value function $Q\left(s,a\right)$ indicates how good action a is given the state s. When an action a is taken at state s, a reward $R(s,a)$ is returned and then the value function is updated as

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a} Q(s',a) - Q(s,a)\right],$$

where α refers to the learning rate $(0 < \alpha \leq 1)$, γ represents the discount factor $(0 \leq \gamma \leq 1)$, and s' is the next state from state s by taking action a. Parameter α determines to what extent the new information should override the old information, and γ determines the importance of future rewards [13]. The basic idea of Q-learning is to update $Q\left(s,a\right)$ via a trial-and-error process, during which actions are selected by a certain rule. The ϵ-greedy $(0 \leq \epsilon \leq 1)$ method is an action selection rule, which aims to take a trade-off between exploitation and exploration. At any state s, the method selects action $\arg\max_a Q(s,a)$ with probability $(1-\epsilon)$ (exploitation), or selects randomly from among all the actions, independently of the Q-value, with probability ϵ (exploration) [13].

The value function $Q(s,a)$ is constantly updated in a series of episodes. An episode is one simulation from the initial state of the environment until the termination state is reached. Based on a well-trained $Q(s,a)$, an optimal action given a state s will be $\arg\max_a Q(s,a)$.

III. Q-learning in train timetable rescheduling

In this section, we first introduce how to represent the railway environment and ensure safe train operations by different headways in Section III-A. Then, we explain how to simulate the railway environment, and the key elements of Q-learning required per simulation in Section III-B.

A. Railway environment representation

A railway environment is represented by stations connected by open tracks as shown in Fig. 1, which is an example of a double-track railway line. Inspired by [12], we define a station as a resource, which consists of at least one track. A track of a station is assumed to be occupied by at most *one* train at a time regardless of the direction from which this train comes. For an open track segment, we define each open track in the segment as a resource. We consider a double-track railway line so that each open track is uni-directional, which means that it can be occupied only by trains running in a specific direction. For example in Fig. 1, the lower track of segment a-b is for the upstream trains only, while the upper track is for the downstream trains only. A track of a segment can be occupied by *multiple* trains simultaneously if these trains are separated by safety distances according to the implemented block signalling system.

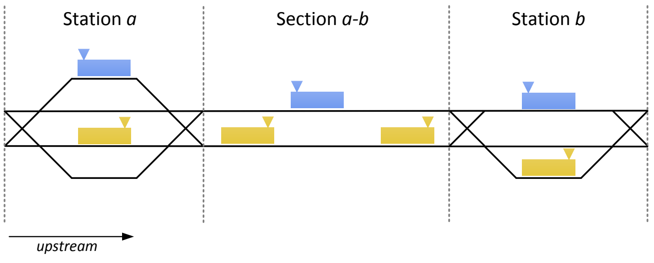

Fig. 1. Example of a double-track railway line

A resource r is associated with multiple attributes, which are introduced in Tab. I. The attributes in the upper part of Tab. I are *fixed*, whereas the attributes in the lower part are *variable*.

For a station resource r_s, the availability is determined by $y(r_s)$, of which each entry indicates the earliest time when a track of r_s will become available. Each track of r_s is assumed to be bi-directional, $d(r_s) = both$. The size of $y(r)$ is equal to the number of tracks in r_s. Each entry of $y(r_s)$ is initialized with value 0 representing that the corresponding track is ready to be used, and when a train starts occupying the track the value will be updated to the expected departure time of this train from this track plus a minimum headway $h_{d,a}$, after which another train can arrive at the same track. In other words, a departure-arrival headway $h_{d,a}$ is enforced between consecutive trains, which will occupy the same track of a station. If an entry of $y(r_s)$ is associated with a time earlier than the current system time and no train will occupy it at that moment, then its value will be updated to 0 again indicating that the corresponding track is available.

TABLE I

ATTRIBUTES OF A RESOURCE r

Attribute	Description
$u(r)$	Index of resource r, which is a unique number assigned to each station or section, $u(r) \in \{1, \dots, n\}$
$p(r)$	The type of resource, which is either a station or a section.
$d(r)$	The possible train directions of r, $d(r) \in \{up, down, both\}$
$n(r)$	The number of tracks in r
$y(r)$	A vector, of which entry i indicates the earliest time when track i of r will become available
$z(r)$	A list, of which element i indicates the train that is currently occupying track i of r

TABLE II

TYPES OF HEADWAYS CONSIDERED IN THIS PAPER

Headway	Description
$h_{d,a}$	The minimum headway between the departure of a train and the arrival of another train, which occupy the same track of a station
$h_{d,d}$	The minimum headway between the departure times of two consecutive trains at the same station. The departure time of a train at a station corresponds to the time of the train entering the next open track from this station.
$h_{a,a}$	The minimum headway between the arrival times of two consecutive trains at the same station. The arrival time of a train at a station corresponds to the time of the train leaving from the previous open track to this station.

For an open track resource r_o, the availability is determined by $y(r_o)$ that consists of only one entry in this case. Note that this paper considers a double-track railway line, so each open track r_o is uni-directional indicating that it can be occupied only by trains running in a specific direction, $d(r_o) = up$ or $d(r_o) = down$. Nonetheless, our method can be easily extended to a mixed-track railway line with few modifications. The value of $y(r_o)$ is initialized to 0 indicating that the open track is ready to be used. When a train enters this open track, $y(r_o)$ will be updated to the entering time (i.e. the departure time from the upstream station relative the open track) plus a minimum headway $h_{d,d}$, after which another train can enter the same track. A minimum headway ($h_{d,d}$ or $h_{a,a}$) is defined as the minimum time separation between the departure/arrival times of two successive trains at a station [15]. To prevent overtaking in an open track (in case of extended running times), a minimum headway $h_{a,a}$ is enforced between two consecutive trains that will arrive at the next station from the same open track. Every time a train t enters an open track, its expected departure time π_t from this track (i.e. the arrival time at the downstream station relative to the open track) is compared to the expected departure time $\pi_{t'}$ of the previous train t' that entered the same open track earlier and is still occupying this track. If $\pi_t - \pi_{t'} < h_{a,a}$, then $\pi_t \leftarrow \pi_{t'} + h_{a,a}$ is performed to respect the minimum arrival-arrival headway. Otherwise, no update will happen to π_t. Note that there is no need to compare a train t to its previous train t' that has already left the open track when train t enters the track. Tab. II lists the headways considered in this paper to ensure safe train operations.

Attribute $z(r_o)$ is initialized as an empty list for an open track resource r_o. Every time a train enters the open track resource r_o, $z(r_o)$ will be added with one element indicating the corresponding train number (a unique number for each train). When a train leaves the open track resource, its train number will be removed from $z(r_o)$. For a station resource r_s, $z(r_s)$ is initialized as a list consisting of $n(r_s)$ elements with value 0. Here, $n(r_s)$ represents the number of tracks in r_s. An element of $z(r_s)$ will be updated when a train arrives at the corresponding track, and will return to value 0 when the train departs from the track.

B. Railway environment simulation

An event-based simulation method is used to simulate the railway environment. An event e is a departure or arrival of a train at a station. Event e is associated with multiple attributes, which are introduced in Tab. III. The attributes in the upper part of Tab. III are *fixed*, while $x(e)$ and $\Delta(e)$ are variables representing respectively the rescheduled time and the delay of event e. The value of $x(e)$ is initialized as $o(e) + \theta(e)$, and $\Delta(e) = x(e) - o(e)$, in which $o(e)$ is the original scheduled time of event e and $\theta(e) \geq 0$ is the initial delay of event e given at the beginning of the simulation. Note that for an arrival event e the resource $r(e)$ to be occupied when e occurs corresponds to station $s(e)$, while for a departure event e its $r(e)$ corresponds to the downstream section relative to station $s(e)$.

TABLE III

ATTRIBUTES OF AN EVENT e

Attribute	Description
$p(e)$	The type of event e, $p(e) \in \{arr, dep\}$
$t(e)$	The train corresponding to event e
$s(e)$	The station corresponding to event e
$r(e)$	The resource to be occupied by train $t(e)$ when event e occurs
$o(e)$	The original scheduled time of event e
$\theta(e)$	The initial delay of event e
$x(e)$	The rescheduled time of event e
$\Delta(e)$	The delay of event e

The set of events is denoted by E. At each simulation *step*, the event e^* with the earliest rescheduled time will be selected,

$$e^* = \arg\min_{e} \{x(e), e \in E\} \tag{1}$$

and an action will be selected by the Q-agent to e^*.

Action set

The agent has an action set $\{0, 1\}$. Action $a = 0$ means that the agent decides *not* to implement event e^* at time $x(e^*)$ but instead delay its occurrence by a fixed λ min. Namely, $x(e^*) \leftarrow x(e^*) + \lambda$. Action $a = 1$ means that the agent decides to implement e^* at time $x(e^*)$. If this action is implementable, then e^* is removed from the event set E. Action $a = 1$ may *not* be implementable if $x(e^*) >$

$\min\{y(r'), r' = r(e)\}$, which indicates that none of the tracks in resource r' is available to receive train $t(e^*)$ at time $x(e^*)$. In this case, the simulation terminates. Here, r' is the resource $r(e)$ to be occupied by event e when e occurs. After each type of action ($a = 0$ or $a = 1$), the variable attributes of the resources and of the events in E will be updated accordingly. For example, if event e^* is delayed for λ min, then the rescheduled times and delays of the following events corresponding to the same train $t(e^*)$ will be updated before starting the next simulation step. Suppose e is one of the following events of train $t(e^*)$, then its rescheduled time $x(e)$ will be updated to the earliest time it could occur, which is counted as $x(e^*)$ plus the shortest time needed from event e^* to event e if this earliest time is later than $o(e)$. Otherwise, $x(e)$ will not be updated since earlier departures/arrivals are not allowed.

Reward function

A penalty -1 is given for an action $a = 0$ since it delays a train arrival/departure. A reward +1 is given for an action $a = 1$ if it is implementable (i.e. no operational conflict). Otherwise, action $a = 1$ is given with a penalty -10.

States

A state s of the railway environment is defined as

$$s = \left\{ \tilde{\Delta}(e^*), c(r_1), \ldots, , c(r_n), r(e^*) \right\}, \tag{2}$$

in which e^* is the current event as defined in Equation (1), $\Delta(e^*)$ is the delay of event e^*, $c(r_i)$ refers to the congestion level of resource r_i, $i \in \{1, \ldots, n\}$, and $r(e^*)$ is the resource to be occupied by train $t(e^*)$ when e^* occurs. $r(e^*) \in \{r_1, \ldots, r_n\}$. We define $\tilde{\Delta}(e^*) = \Delta(e^*)$ if $\Delta(e^*) \leq D$, where D is the largest delay considered. Otherwise, $\tilde{\Delta}(e^*) = D$. The congestion level of a resource is defined respectively for a station resource and a section resource as follows.

For a *station* resource r_s, the congestion level $c(r_s)$ is 0 if all tracks in r_s are available, 1 if one track in r_s is not available, or 2 if at least two tracks in r_s are not available. For an *open track* resource r_o, the congestion level $c(r_o)$ is 0 if r_o is not occupied by any train, 1 if r_o is occupied by one train, or 2 if r_o is occupied by at least two trains. Recall that in Section III-A we explained that a track of a resource is available or not depends on the corresponding entry in $y(r)$, which indicates the earliest time that the track will become available. Only if the earliest available time of a track is smaller than $x(e^*)$, then the track is considered as available at the current simulation step.

The size of a state s is relevant to the number of resources considered. In the extreme case, all resources included in the railway environment are considered, which may lead to a large state size causing memory issues. Therefore in a state vector s, only n resources are considered, which include the resource that is currently occupied by train $t(e^*)$, and the next $n-1$ resources to be occupied by train $t(e^*)$. Note that the route for each train is assumed to be fixed so that the next n resources are always known. If the total number of the next resources m is smaller than $n-1$ ($m < n-1$), then the $(n-(m+1))$ resources behind train $t(e^*)$ will be considered in state vector s to ensure s has n resources in total.

The total number of states $|S|$ is $(D+1) \times 3^n \times n$, where D is the largest delay considered (note that a delay can be 0), 3 refers to the number of congestion levels of a resource, and n is the number of considered resources. $|S|$ is *not* relevant to the scale of the railway environment and the total number of trains. In Section IV, we will investigate the impact of n on the performance of the Q-learning method in terms of solution quality.

A simulation of the railway environment is initialized with all trains waiting at their origins, and will terminate when all trains reached their destinations (corresponding to an empty E) *or* if an action $a = 1$ is not implementable due to a conflict. A simulation is one training episode for the agent to learn the state-action value function $Q(s, a)$. The agent are trained with sufficient episodes to obtain a good $Q(s, a)$.

IV. Case study

We use a double-track railway line from the Netherlands to investigate the performance of the proposed reinforcement learning-based timetable rescheduling method. Fig. 2 shows the track layout of the considered railway line, as well as the train lines, which are operating half-hourly in each direction.

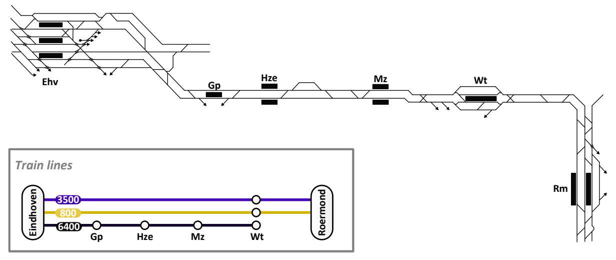

Fig. 2. The track layout and line plan of the considered railway

A. Parameter settings

For the Q-learning method, we set the learning rate α to 1, and the discount factor γ to 0.9. The state-action value function $Q(s, a)$ is initialized with $Q(s, 0) = -1$ and $Q(s, 1) = 1$, $\forall s \in S$. This is because our objective is minimizing train delays so that moving a train forward ($a = 1$) is generally preferred over holding a train ($a = 0$) unless it leads to a conflict. A good initial $Q(s, a)$ helps to speed up the convergence of a goal-directed reinforcement learning [14]. For the ϵ-greedy action selection rule, the value of ϵ is initialized as 0.5, which is applied with a decay rate of 0.9 in each following episode. In that sense, we favour exploitation when gaining more training experience. The total number of training episodes is set to 50.

The value of λ is set to 0.5 min, which means that an action $a = 0$ is to delay an event e for 0.5 min longer than its current rescheduled time $x(e)$. All types of minimum headways, $h_{d,a}$, $h_{a,a}$ and $h_{d,d}$, are set to 3 min.

B. Influence of the state vector on solution quality

Recall Equation (2), in which a state vector s is defined. Experiments are designed to investigate how the elements in s impact the solution quality. First, we consider whether including $\tilde{\Delta}(e^*)$ in state s will have a difference in solution quality. Recall that $\tilde{\Delta}(e^*)$ indicates the delay of event e^*. We also investigated the impact of the number of resources n considered in s. In this section, we consider a single initial delay that the departure of the 2nd upstream train from line 3500 (in purple) is initially delayed for 8 min at its origin station Roermond (Rm). This departure event is highlighted with a red circle in Fig. 3 and Fig. 4.

TABLE IV

RESULTS WITH DIFFERENT STATE REPRESENTATIONS

Case	$\tilde{\Delta}(e^*)$	n	Total delay [min]	Case	$\tilde{\Delta}(e^*)$	n	Total delay [min]
I.1	-	2	273	II.1	[0, 10]	2	169
I.2	-	3	273	II.2	[0, 10]	3	140
I.3	-	4	193	II.3	[0, 10]	4	116
I.4	-	5	157	II.4	[0, 10]	5	116
I.5	-	6	157	II.5	[0, 10]	6	116
I.6	-	7	123	II.6	[0, 10]	7	114

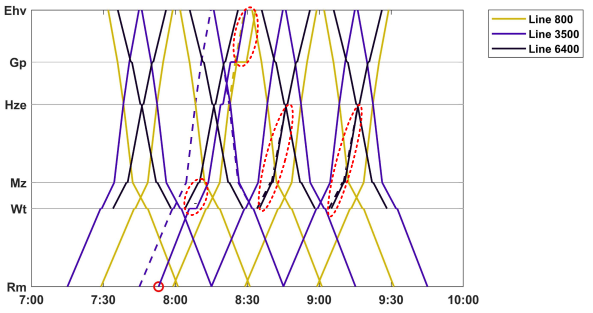

Fig. 3. Rescheduled timetable of case I.6, in which state s *excludes* delay element $\tilde{\Delta}(e^*)$ and considers $n = 7$ resources

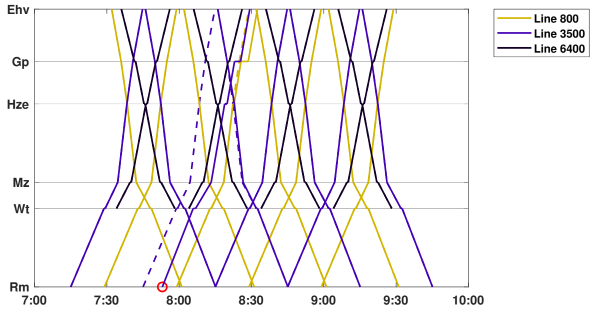

Fig. 4. Rescheduled timetable of case II.6, in which state s includes the delay element $\tilde{\Delta}(e^*) \in [0, 10]$ and considers $n = 7$ resources

Tab. IV shows the total train delay of the final solution (after all training episodes) obtained by the Q-learning method under different settings of state vectors. The left part of Tab. IV shows the results of excluding $\tilde{\Delta}(e^*)$ in s with different n, and the right part of Tab. IV shows the results of including $\tilde{\Delta}(e^*)$ in s with different n, in which $[0, 10]$ represents the range of delay values. Tab. IV indicates that the representation of a state affects the solution quality in terms of the total train delay. More resources (larger n) considered in a state s leads to a better rescheduled timetable with less train delays. Also, including the delay $\tilde{\Delta}(e^*)$ of the current event e^* results in a better rescheduling solution.

Fig. 3 shows the rescheduled timetable of case I.6 that does *not* includes $\tilde{\Delta}(e^*)$ and considers 7 resources in a state. The solid lines represent the rescheduled train services, and the dashed lines refer to the original scheduled train services that are delayed in the rescheduled timetable. The initial train delay is highlighted with a red solid circle. The train services which experienced *unnecessary* delays in the obtained rescheduling solution are highlighted with red dotted ellipses. For example in section Wt-Mz the time separation between an upstream train from line 6400 (in black) and an upstream train from line 3500 (in purple) is larger than the minimum headway required, and this also happened to two upstream trains running from station Gp to station Ehv. There are also two upstream trains from line 6400 (in black) delayed when departing from station Wt, which were unnecessary from the perspective of required headways.

Compared to Fig. 3, a better rescheduled timetable with less delays is shown in Fig. 4, which corresponds to case II.6 that includes $\tilde{\Delta}(e^*)$ and considers 7 resources in a state. In Fig. 4, all secondary delays are necessary in order to respect operational requirements.

C. *Influence of the state vector on solution convergence*

We computed the total reward obtained from a simulation episode, which are the sum of rewards/penalties of all actions taken in this episode. For each case of Tab. IV, the total reward per episode is shown in Fig. 5 and Fig. 6.

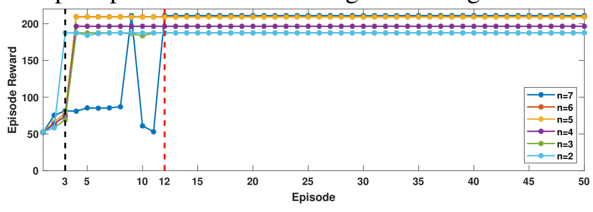

Fig. 5. Episode rewards of cases I.1-I.6, which *exclude* the delay element $\tilde{\Delta}(e^*)$ in a state s

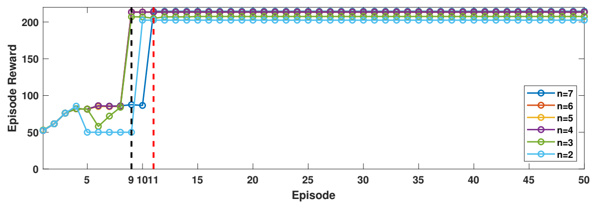

Fig. 6. Episode rewards of cases II.1-II.6, which *include* the delay element $\tilde{\Delta}(e^*)$ in a state s

Fig. 5 is the results of cases I.1-I.6, which *exclude* the delay element $\tilde{\Delta}(e^*)$ in a state s. It can be seen that when considering $n = 2$ resources in state s no solution improvements were found after 3 episodes, whereas when considering $n = 7$ resources more episodes (12 episodes) were required for obtaining a *stable* solution. These are highlighted by the black and red dashed lines, respectively. Nevertheless, when excluding $\tilde{\Delta}(e^*)$ in a state s the solution converges around 4 episodes in most of the corresponding cases.

Fig. 6 is the results of cases II.1-II.6, which *include* the delay element $\tilde{\Delta}(e^*)$ in a state s. In all of these cases, at least 9 episodes were needed for solution convergence. This is because of the increasing state space by including $\tilde{\Delta}(e^*)$. Nevertheless, after 11 episodes all cases obtained stable solutions. The black and red dashed lines highlight respectively the smallest and largest numbers of training episodes required for obtain stable solutions among cases II.1-II.6.

D. *Multiple initial delays*

In this section, we set up another two delay scenarios, which have respectively 2 and 3 initial delayed train events. We trained the Q-agent in each delay scenario with the same state representation as in case II.6 shown in Tab. IV. The rescheduled timetables for the two delay scenarios are shown respectively in Fig. 7, and Fig. 8, where the initial delayed events were highlighted with red circles. All secondary delays in these scenarios are necessary to avoid operational conflicts.

The scenarios with 2 and 3 initial delays needed respectively 12 and 30 training episodes to find stable solutions. This indicates that with more number of initial delays considered, the Q-agent needs more training episodes for solution convergence.

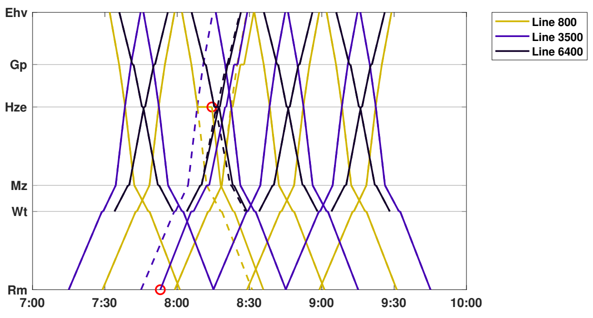

Fig. 7. Rescheduled timetable with 2 initial delays

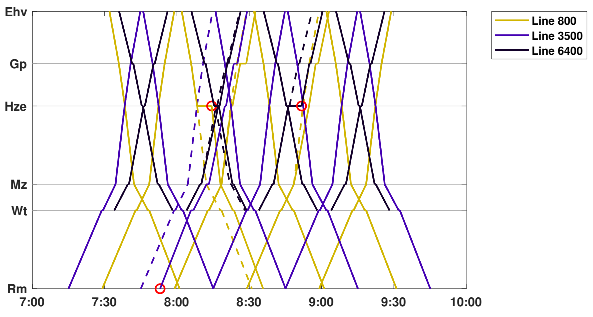

Fig. 8. Rescheduled timetable with 3 initial delays

V. Conclusions

This paper employs a reinforcement learning method, more specifically Q-learning, to realize railway timetable rescheduling. Multiple types of headways are considered in this method to ensure that the rescheduled timetables respect the operational requirements. Numerical experiments were performed to investigate the influence of the state representation on solution quality and solution convergence. It was found that an effective state representation of the railway environment helps the Q-agent to learn the optimal action to be taken for each train event. An effective state representation considers more elements of the environment, and thus increases the state space and slows down the solution convergence. Nevertheless, in all cases with one single initial delay, at most 12 training episodes were needed to find stable solutions. We tested the proposed Q-learning method also to scenarios with multiple initial delays, under which more training episodes (at most 30) were required for solution convergence compared to the scenarios with a single initial delay.

In future, we will deal with a larger railway network with more stations and trains. Besides, we will employ extensive historical delay data from the actual railway operations to train the Q-agent so that it can be more experienced to handle various situations with better rescheduling solutions. Another direction will be extending the reinforcement learning-based timetable rescheduling method for major disruptions [16].

References

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

[2] F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo, "A tabu search algorithm for rerouting trains during rail operations," *Transportation Research Part B: Methodological*, vol. 44, no. 1, pp. 175-192, 2010.

[3] F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo, "Dispatching and coordination in multi-area railway traffic management," *Computers & Operations Research*, vol. 44, pp. 146-160, 2014.

[4] J. Törnquist, and J. A. Persson, "N-tracked railway traffic rescheduling during disturbances," *Transportation Research Part B: Methodological*, vol. 41, no. 3, pp. 342-362, 2007.

[5] J. T. Krasemann, "Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances," *Transportation Research Part C: Emerging Technologies*, vol. 20, no. 1, pp. 62-78, 2012.

[6] S. P. Josyula, J. T. Krasemann, and L. Lundberg, "A parallel algorithm for train rescheduling," *Transportation Research Part C: Emerging Technologies*, vol. 95, pp. 545-569, 2018.

[7] S. Van Thielen, F. Corman, and P. Vansteenwegen, "Considering a dynamic impact zone for real-time railway traffic management," *Transportation research part B: methodological*, vol. 111, pp. 39-59, 2018.

[8] L. Ning, Y. Li, M. Zhou, H. Song, and H. Dong, "A deep reinforcement learning approach to high-speed train timetable rescheduling under disturbances," In *the 22nd IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 3469-3474, Oct, 2019.

[9] T. Ghasempour, and B. Heydecker, "Adaptive railway traffic control using approximate dynamic programming," *Transportation Research Part C: Emerging Technologies*, 2019.

[10] T. Ghasempour, G. L. Nicholson, D. Kirkwood, T. Fujiyama and B. Heydecker, "Distributed Approximate Dynamic Control for Traffic Management of Busy Railway Networks," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[11] 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*, vol. 86, pp. 250-267, 2016.

[12] H. Khadilkar, "A scalable reinforcement learning algorithm for scheduling railway lines," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 2, pp. 727-736, 2019.

[13] R. S. Sutton, and A. G. Barto, *Reinforcement learning: An introduction*, 2nd Edition, Cambridge, MA, USA: MIT Press, 2018.

[14] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning," In *International Conference on Artificial Neural Networks*, Springer, Berlin, Heidelberg, pp. 840-849, 2006.

[15] R.M.P. Goverde, F. Corman, and A. D'Ariano, "Railway line capacity consumption of different railway signalling systems under scheduled and disturbed conditions," *Journal of Rail Transport Planning & Management*, vol. 3, no. 3, pp. 78-94, 2013.

[16] Y. Zhu, and R.M.P. Goverde, "Railway timetable rescheduling with flexible stopping and flexible short-turning during disruptions," *Transportation Research Part B: Methodological*, vol. 123, pp. 149-181, 2019.