Transportation Research Part C 183 (2026) 105453

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Learning to reschedule platforms: A graph neural network based deep reinforcement learning method for the train platforming and rescheduling problem ⋆

Hongxiang Zhang [a], Andrea D'Ariano [b], Yongqiu Zhu [c], Yaoxin Wu [d], Liuyang Hu [a], Gongyuan Lu [a,e,f,*]

[a] School of Transportation and Logistics, Southwest Jiaotong University, 610031, Chengdu, China
[b] Department of Civil, Computer Science and Aeronautical Technologies Engineering, Roma Tre University, 00146, Rome, Italy
[c] Department of Transport and Planning, Delft University of Technology, 2628 CN, Delft, The Netherlands
[d] Department of Industrial Engineering & Innovation Sciences, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands
[e] Comprehensive Transportation Key Laboratory of Sichuan Province, 610031, Chengdu, China
[f] National United Engineering Laboratory of Integrated and Intelligent Transportation, 610031, Chengdu, China

ARTICLE INFO

Keywords:
Train platforming
Train rescheduling
Neural combinatorial optimization
Graph neural network
Deep reinforcement learning

ABSTRACT

The train platforming schedule is the crucial plan for guiding trains to travel through a railway station without spatial and temporal conflicts. When trains are delayed in arriving at the station due to disturbances or disruptions, it raises the Train Platforming and Rescheduling Problem (TPRP), one of the hot topics in railway traffic management. It focuses on allocating platforms and time slots for trains to reduce delays and ensure operational efficiency in a station. This paper introduces a novel graph neural network based deep reinforcement learning method to address this problem, named Learning to Reschedule Platforms (L2RP). We formulate the solving process of TPRP as a customized Markov decision process. Meanwhile, we integrate a microscopic discrete-event train operation simulation model to serve as the agent exploration environment, which provides states, executes actions, and completes transitions. Then, we design a hybrid graph neural network based policy network to derive high-quality actions under each graph encoded state. The policy network is trained with the reward function designed to minimize total train knock-on delays and platform changes. The experiments on real-world instances show that the proposed L2RP method can produce high-quality solutions for instances of various scenarios within stably short solving times.

1. Introduction

The **T**rain **P**latforming and **R**escheduling **P**roblem (**TPRP**) is a variant of the train platforming problem (TPP), which is a widely investigated topic of train operation planning in railway stations.

⋆ Given their role as Associate Editor, Andrea D'Ariano had no involvement in the peer review of this article and had no access to information regarding its peer review. Full responsibility for the editorial process for this article was delegated to another journal editor.

* Corresponding author.
E-mail addresses: hongxiang@my.swjtu.edu.cn (H. Zhang), andrea.dariano@uniroma3.it (A. D'Ariano), y.zhu-5@tudelft.nl (Y. Zhu), y.wu2@tue.nl (Y. Wu), huliuyang@swjtu.eud.cn (L. Hu), lugongyuan@swjtu.edu.cn (G. Lu).

https://doi.org/10.1016/j.trc.2025.105453
Received 22 March 2025; Received in revised form 7 November 2025; Accepted 10 November 2025
Available online 22 November 2025

The main purpose of TPP is to simultaneously schedule arrival and departure times and assign platforms for trains, ensuring that trains operate punctually, safely, and efficiently in a railway station. TPP variants can be categorized into three aspects according to the planning stage: strategical, tactical, and operational levels (Sels et al., 2014).

At the operational level, platform assignments must be made in real time, which is a critical task when trains are delayed due to disturbances or disruptions. These deviations probably result in route conflicts or platform contention among trains. To ensure conflict-free train operations and minimize train delays, trains are required to be quickly and effectively rescheduled to new time slots and reassigned to new platforms. This dynamic and reactive nature of the problem, where adjustments must be made rapidly, is the core focus of the TPRP addressed in this paper.

In the context of TPRP, both solution quality and computational efficiency are of paramount importance. On the one hand, arbitrary platform assignments and inefficient train schedules can further exacerbate operational challenges, including increased route conflicts, prolonged train delays, and aggravated station congestion. On the other hand, computational efficiency is crucial for generating platforming and rescheduling decisions. Even if some preventive measures, such as robust timetabling or dispatcher oversight, could have been conducted before deviation occurs, real-time resolution remains indispensable, as delays can escalate rapidly across trains in a network. This is particularly critical to face major disruptions, where train delays cannot be absorbed by the timetable's buffer time. In such cases, the faster feasible platforming and rescheduling solutions are provided and executed, the smaller the number of affected trains and the duration of disruptions become.

However, it is extremely challenging to achieve high-quality solutions of TPRP under strict time restrictions. Although mathematically high-quality solutions can be attained with coarse-grained models of train operations, fine-grained modeling (e.g., track sections and fine time units) enables more close-to-reality and faithful solutions by explicitly capturing operational details and resource constraints. Yet, such microscopic details significantly increase model complexity and incur substantial computational costs of TPRP, undermining its applicability in real-time scenarios.

Therefore, this paper aims to leverage microscopic modeling, produce high-quality solutions, and ensure computational efficiency in addressing TPRP, so that reliable train schedules and platform assignments can be provided in a timely manner to minimize train delays.

Existing research on TPRP and its variants are predominantly based on mathematical programming and heuristic algorithms (Lusby et al., 2011). Although these approaches often produce high-quality solutions, achieving both fine spatial and temporal precision in the mathematical models and maintaining high computational efficiency still remains as a significant challenge, particularly when the problem is scaled to hundreds of trains or is specialized to TPRP (Zhang et al., 2023a, 2025b; Kang et al., 2019). Furthermore, while current studies typically focus on random delays, they often lack generalization or validation for real-world disruption scenarios (Lu et al., 2022; Teng et al., 2023; Zhang et al., 2020b). These limitations hinder the practical application of traditional optimization methods in real-world train operation management.

In recent years, the deep reinforcement learning (DRL) technique has revealed promising capability in solving classical combinatorial optimization problems (Wu et al., 2021; Smit et al., 2024; Wang et al., 2024). DRL methods model the solving process of a specific optimization problem as a Markov decision process (MDP), including states, actions, and rewards. An environment is set to provide states, execute actions, and generate rewards. Hence, instead of heavy manual work on refining modeling granularity in mathematical formulation, precision can be ensured by the environment depicting the problem. A simulation is usually utilized as the environment when solving industrial control and management problems (Ye et al., 2019; Reda et al., 2020; Bigi et al., 2024; Bosi et al., 2024). Meanwhile, graph neural networks (GNNs) have been proven to enhance policy learning in DRL methods by parameterizing the policies with informative graph encoded data in DRL frameworks for combinatorial optimization problems. Using trained DRL agents, this learning fashion shows high solving efficiency, comparable performance to specialized algorithms, and good generalization ability in various scenarios (Cappart et al., 2023; Zhang et al., 2025a; Lin et al., 2024).

Therefore, in this paper, we propose a novel graph neural network based deep reinforcement learning method to solve the TPRP, namely **L**earning **to R**eschedule **P**latforms (**L2RP**). Specifically, we first formulate the process of solving the TPRP as an MDP, in which a state is encoded to a graph structure with three node sets representing inbound trains, berthing tracks, and routes, respectively. To reschedule trains and assign platforms, the action space in the MDP model contains dispatching actions associated with routes and a postponing action. Simultaneously, we design a transition mechanism that can synchronously designate the decision moment and the decision object train for the next state according to four situations of computing succeeding decision moments. To implement the above MDP model, a microscopic discrete-event simulation model of train operations in the railway station is developed, which can generate masked action space, execute selected actions (i.e., decisions) for trains, and complete state transitions. It is integrated into the DRL framework by serving as the agent exploration environment. Then, we design a graph neural network based policy network to be the decision-making agent, which consists of two parts. Its first part is a hybrid architecture of graph neural networks, combining Graph Attention Networks and Graph Isomorphism Networks, named hAI-GNN. This hybrid network learns node embeddings over the graph encoded state. The other part of the policy network, the action reasoning network, leverages the embeddings to compute action selection probabilities under each state. Finally, the proximal policy optimization (PPO) algorithm is used to train the policy network according to the reward function of minimizing train knock-on delays and platform changes. The trained decision-making agent is able to deliver favorable solutions for TPRP instances with various problem sizes and delay scenarios in very short solving times.

Based on the above implementations, the contribution of this work can be summarized as follows:

- We propose a novel deep reinforcement learning framework for the train platforming and rescheduling problem. In this framework, a Markov decision process with graph encoded states and a lightweight action space is formulated, and a graph neural network

based policy network is designed. This framework captures complex interdependencies among trains, platforms, and routes, enabling intelligent decision-making of platforming and rescheduling.
- The proposed deep reinforcement learning framework integrates a microscopic simulation model of train operations in the station, which employs a fine-grained modeling of spatial and temporal resources. This integration ensures solution precision and enhances applicability to real-world applications.
- Our approach can produce high-quality solutions with consistently short solving times, demonstrating strong potential for real-time application in large stations. The solution quality is competitive with established heuristic benchmarks, achieving near-optimal results.
- The framework demonstrates strong performance when generalized to unseen timetables and severe line disruption scenarios. It delivers solutions that can recover train operations significantly faster than conventional heuristic rules, validating its utility under diverse uncertainty.

The remaining part of this paper is organized as follows. Section 2 briefly introduces the background knowledge of TPRP, and then reviews the related works on train platforming and the application of DRL to train traffic management. Section 3 describes the TPRP in detail and introduces a concept used to model route conflicts. Then, the proposed L2RP method is elaborated in Section 4, including the MDP model and the policy network. In Section 5, we report experiment results to show the solution quality, efficiency, and generalization capability of the L2RP method. The last section concludes this study.

2. Background

In this section, we first briefly introduce the basic knowledge of train operation management in a railway station, expecting to provide convenience for readers to understand terms in the subsequent literature review. Then, we review existing research regarding the train platforming problem. Finally, we recap studies that apply reinforcement learning methods in train scheduling.

2.1. Platform, route, route conflicts, and interlocking mechanism

Before entering the technological part, we first introduce some necessary conceptions regarding TPRP using an illustrative example of a railway station, which has 6 platforms, 2 entry points, and 2 exit points as shown in Fig. 1. A platform has only one corresponding berthing track. 'Berthing track' and 'platform' are two equivalent resources from the perspective of train dispatching. 'Platforming' means allocating berthing tracks for trains to dwell in the station.

Trains enter and leave the station via routes. More specifically, a receiving route connects an entry point and a berthing track. An inbound train travels through an assigned receiving route to arrive in the berthing track. Similarly, an outbound train requires a departure route connecting the berthing track and the exit point to leave the station. As shown in Fig. 1, a route contains a sequence of physical resources, i.e., tracks and switches, collectively called 'sections'.

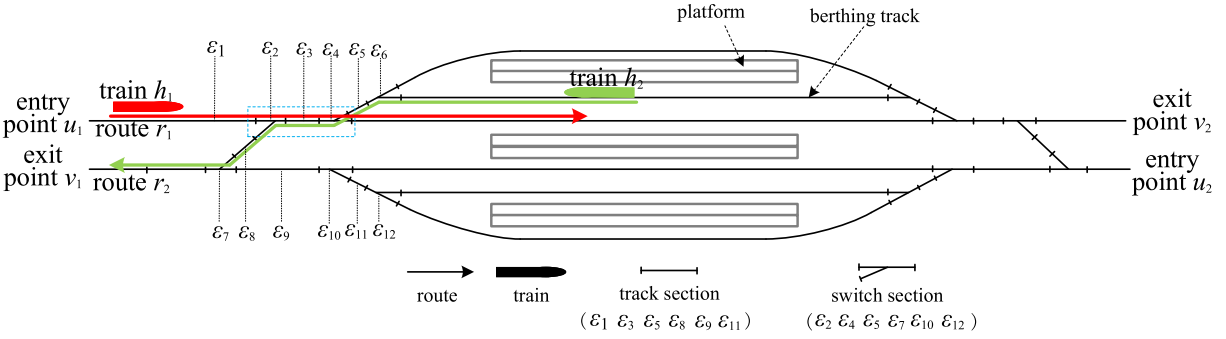


Fig. 1. An example of railway station layout.

Route conflicts may arise in practical train operations if two routes share one or more sections (Zhang et al., 2024). As illustrated in Fig. 1, sections ϵ_2, ϵ_3, and ϵ_4 are components of both the receiving route r_1 and departure route r_2. When the receiving route r_1 is claimed for inbound train h_1, route r_1 will be locked, which means prohibiting any sections in this route from being used for any other trains. In this case, if the route r_2 is required to be claimed for the outbound train h_2 simultaneously, train h_1 and h_2 have a route conflict since the shared sections ϵ_2, ϵ_3, and ϵ_4 are still occupied by train h_1. Route conflicts are prevented using interlocking mechanisms in the real-world railway operation. The route-lock-section-release interlocking mechanism is widely used to resolve route conflicts (Corman et al., 2009). Under this mechanism, a route will be entirely locked when it is claimed for a train. During the train running process on the route, once the train clears a section, the section will be released immediately and independently. Hence, sections in a claimed route will be released sequentially along with the train traversing the route. For example, to avoid the route conflicts between route r_1 and route r_2 in Fig. 1, claiming route r_2 for train h_2 must wait until train h_1 clears the last shared section ϵ_4 in route r_1.

Route conflicts directly influence the feasibility of train platforming. Even if a berthing track is idle, it can still not be assigned for an inbound train if the receiving route connecting it and the entry point has conflicts with other routes. Therefore, when make platforming and rescheduling decisions for trains in the railway station, the receiving and departure routes must be arranged without route conflicts based on the interlocking mechanism. It is the main focus of the TPRP. In the next section, we review literature related to this topic.

2.2. Literature review of train platforming problem

In recent years, the train platforming problem has been a research hotspot in the field of train operation management. As stated in Lusby et al. (2011) and Sels et al. (2014), from the perspective of research stage, studies about TPP can be categorized into three aspects, i.e., strategical, tactical, and operational level. The strategical level studies concern the layout design and capacity of a railway station. The tactical level of the TPP, where most studies of the TPP and its variants are located, usually aims to schedule given trains in the fixed railway station. At the operational level, real-time train delays may occur due to unexpected events, leading to the train platforming plan resulting from the tactical level not being feasible. In this case, routes and platforms are required to be decided in real time, so TPP turns to its variant Train Platforming and Rescheduling (Lu et al., 2022), which is the most challenging computationally. Considering that detailed railway station information is given in our work and we reschedule trains at the operational level, we here concentrate on previous studies that model the TPP and its variants at the tactical or operational level, and refer readers interested in strategical studies of TPP to Zwaneveld et al. (1996, 2001)

The TPP studies at the tactical level make platforming decisions (i.e., schedule receiving and departure routes and assign platforms) for trains in a given timetable. Primary train delays are not considered, but train arrival and departure times could be shifted by scheduling decisions if necessary. Billionnet (2003) is the first to use integer programming to solve the TPP, where the integer programming model is equivalent to the graph coloring problem. In this study, the assignment of a train to a platform also uniquely determines the receiving and departure routes for the train. In Carey and Carville (2003), trains can be delayed to generate a feasible solution. The scenario that more than one train can dwell on the same berthing track is considered. The route conflict avoidance in the developed mathematical model is realized by labeling conflict routes using binary variables and separating trains using headways. Caprara et al. (2011) uses the dummy platforms to ensure the developed mathematical model can always obtain feasible solutions. The slight shifts on the ideal arrival and departure times are allowed. The safety constraints for train paths and platforms are realized simply by time intervals. Similar to this study, Sels et al. (2014) also uses the dummy platforms to hold the overflowed trains. In this study, train arrival and departure times can not be adjusted. Only platform assignment is required to be determined. The route conflict is resolved by setting time separations between route pairs. Zhang et al. (2018) solves the TPP under the similar problem settings, i.e., no time deviation from the given timetable is allowed. By simplifying all receiving and departure have the same length and train running time, the pre-calculated time points and grouped berthing tracks are used to construct conflict-free constraints in the developed 0–1 integer programming. The above studies simplify the modeling of route conflicts in the TPP, enabling acceptable model scales and solving time for relatively long planning periods (e.g., a one-day timetable). If the route details are included to solve TPP from the microscopic perspective, although sophisticated models are developed, the computational cost is sharply increased. A very latest study is Zhang et al. (2025b), in which a two-level space-time network is constructed to capture detailed resource infrastructure information in routes. A two-level Lagrangian Relaxation method is developed to solve the time-space network based nonlinear programming model. Experiments using various time discretization for the time-space network are conducted. Even for the small instance with 4 platforms and 20 trains, if the time precision is discretized to 5 seconds, the proposed method needs 1136 seconds to obtain the solution. A trade-off between the solution quality and the modeling precision has to be considered, so the 15-second time precision and a 30-minute solving time limit are set for large instances. Zhang et al. (2023a) also uses the time-space network to model microscopic resources of route in the TPP. By setting the time precision to 1 minute, the proposed Alternating Direction Method of Multipliers based method requires 894.22 seconds to solve an instance with 40 trains.

The operational level studies of the TPP (i.e., TPRP) also face the granularity challenge, especially considering the real-time solving requirements to cope with train delays. Chakroborty and Vikram (2008) formulates the TPRP as a mixed integer linear programming model to minimize the weighted total train delays and the cost of platform reassignment. The operation safety in the railway station only focuses on the platform and is guaranteed by headways. No route conflicts are considered, while it still needs 10 minutes to solve an instance with 9 platforms and 110 trains. Similarly, Zhang et al. (2020b) also uses headways to separate trains assigned to the same platform. A time-space network based mathematical model and a genetic and simulated annealing hybrid method are developed. The time precision is set to 5 minutes to achieving short solving time (32.92 seconds) for a large instance (11 platforms and 70 trains). Teng et al. (2023) still uses headways in its mathematical model of TPRP to avoid train operation conflicts. The proposed rolling horizon algorithm needs over 15 minutes to solve an instance with 12 platforms and 227 trains. Kang et al. (2019) studies the TPRP with an unfixed timetable from the microscopic perspective. The routes, route conflicts, and safety constraints in the railway station are modeled based on grouped switches. A tailored Simulated Annealing algorithm is proposed to solve the problem. For the instance with 9 platforms and 28 trains, the solving time has already been up to 726 seconds. García-Ródenas et al. (2024) partially models the route occupation by the multi-block section and the headway defined for it, which is a mesoscopic method. A greedy-based heuristic solution approach is developed to solve the TPRP in real time. The solving time of an instance with 10 platforms and 132 trains is reported as 'a few seconds', while the time precision is the proposed method is not clear. Since the number of multi-block section affects the model scale, the performance may degenerate in railway stations with more complex switch areas. Our previous work (Lu et al., 2022) proposes the concept of Degree of Conflict to describe the route conflicts based on sections, enabling microscopic spatial and temporal modeling for the TPRP and testing different interlocking mechanisms flexibly. However, at most 3 trains are

Table 1
The overview of related works.

Publication	Resource modeling precision	Time modeling precision	Problem type	Platform Re-assignment?	Deviations from given timetable?	Line disruption scenarios?	Largest instance
Carey and Carville (2003)	Route	unknown	TPP	✓	✓	✗	12 platforms, 491 trains
Caprara et al. (2011)	Route	1 min	TPP	✗	✗	✗	14 platforms, 237 trains
Dewilde et al. (2013)	Route	unknown	TPP	✓	✓	✗	12 platforms, 90 trains
Sels et al. (2014)	Route	unknown	TPP	✓	✗	✗	14 platforms, 160 trains
Zhang et al. (2018)	Route	unknown	TPP	✓	✗	✗	12 platforms, 234 trains
Meng et al. (2021)	Section	1 min	TPP	✓	✓	✗	4 platforms, 34 trains
Zhang et al. (2023a)	Section	1 min	TPP	✓	✓	✗	12 platforms, 40 trains
Zhang et al. (2025b)	Section	15 sec	TPP	✗	✓	✗	12 platforms, 287 trains
Chakroborty and Vikram (2008)	Route	1 min	TPRP	✓	✓	✗	9 platforms, 110 trains
Zhang et al. (2020b)	Route	5 min	TPRP	✓	✓	✗	11 platforms, 70 trains
Teng et al. (2023)	Route	1 min	TPRP	✓	✓	✗	12 platforms, 227 trains
García-Ródenas et al. (2024)	Route	unknown	TPRP	✓	✓	✗	10 platforms, 849 trains
Kang et al. (2019)	Section	1 min	TPRP	✗	✗	✗	9 platforms, 28 trains
Lu et al. (2022)	Section	5 sec	TPRP	✓	✓	✗	15 platforms, 100 trains
This paper	Section	1 sec	TPRP	✓	✓	✓	15 platforms, 200 trains

set with primary delay among 100 trains, and the solving time still needs a couple of minutes. We use the rolling horizon algorithm proposed in this work as a benchmark for our L2RP method in experiments, since it achieves the finest time units (5 seconds) among the above reviewed research, outputs (near-)optimal solutions, and maintains reasonable solving time. The above research on TPRP typically reports solving times at the level of several minutes. However, realistic train delay scenarios require methods with much higher computational efficiency. As discussed in Cacchiani et al. (2014), Cadarso et al. (2015), García-Ródenas et al. (2024), only minor disturbances can be absorbed by the robust or resilient timetable quickly, whereas major disruptions caused by failing rolling stock, crew shortage, or infrastructure blockage require a new substantial recovery plan, as system buffers are rendered ineffective. Therefore, highly efficient methods capable of providing TPRP solutions within sub-minute or even second-level solving times can empower dispatchers to intervene as early as possible, thereby minimizing the knock-on delays and the overall impact on train operations. This need for high solving efficiency has been recognized in many studies on real-time train dispatching, including TPRP (Pellegrini et al., 2014; Lamorgese and Mannino, 2015; Bettinelli et al., 2017).

An overview of the above reviewed research is provided in Table 1. Generally, whether considering real-time train delays determines the problem type is TPP at the tactical level or TPRP at the operational level. Modeling track resources based on route and using less-refined time units are able to control the model sizes and achieve relatively short solving times with highly specialized heuristic algorithms, especially when facing the real-time solving requirement of handling delayed trains at the operation level. However, such a scarification will undermine the precision of scheduling solutions for trains and the contribution to real-world applications.

Last but not least, the train delay scenarios in current TPRP studies are mostly set by random delays, even if such delays are claimed to be caused by 'disruptions'. Actually, trains delayed due to the disruption may arrive frequently after the disruption ends to resume train operations as soon as possible. It is a totally different scenario from random delays, which leads to new conflicts between delayed trains and originally scheduled trains (Narayanaswami and Rangaraj, 2013; Pineda-Jaramillo et al., 2023). However, the current TPRP studies have not been validated on this train delay scenario, and other existing research about line disruption usually only considers platforms as capacity constraints from the macroscopic perspective of rescheduling trains in railway networks (Zhan et al., 2015; Zhang et al., 2020a; Zhu and Goverde, 2020; Zhang et al., 2023b)).

In summary, using the optimization fashion, it is hard to simultaneously model the TPRP at the microscopic spatial and temporal level and solve the TPRP in a very short solving time. The performance of dealing with train delay scenarios caused by line disruptions in TPRP has yet to be tested. Therefore, to fill these research gaps, this paper proposes a graph neural network based deep reinforcement learning method (i.e., L2RP) to solve the TPRP. Our L2RP method employs a microscopic discrete-event simulation model to be the environment of DRL, modeling train movements and route conflicts based on sections and refining the time precision to 1 second. More significantly, the trained decision-making agent of L2RP can still obtain high-quality solutions very fast under such a microscopic modeling manner. Meanwhile, the trained decision-making agent can be generalized to various timetables and train delay scenarios, maintaining high-quality solutions and increasing solving times linearly with the number of trains.

2.3. Literature review of applying DRL to train scheduling problems

As a promising technique for solving combinatorial optimization problems, DRL has been applied in a minority of studies regarding train scheduling problems (Jusup et al., 2021; Zhang and Zhang, 2023). A pioneer work is Šemrov et al. (2016), which develops a Q-learning based algorithm to optimize the timetable rescheduling on a single-track railway line for minimizing train delays. Aiming

to minimize the total priority-weighted delay for trains on railway lines, Khadilkar (2018) proposes a scalable Q-learning based reinforcement learning method, in which the action space is invariant with the size of the problem instance. Zhu et al. (2020) proposes a reinforcement learning based timetable rescheduling method. Multiple types of headways on the railway line are considered to ensure the rescheduled timetables respect necessary operation requirements. The stations and line segments are simplified to uniform resources with determined capacity. Ying et al. (2020) presents an actor-critic deep reinforcement learning approach for metro train scheduling considering the circulation of limited rolling stock. Dispatch headway, running time between stations, and dwell time at stations are decision variables in the developed Markov decision process, while the capacity of stations is not clearly stated. As a follow-up study, Ying et al. (2022) uses a deep reinforcement learning method to simultaneously solve both the train composition problem and train scheduling problem on a single metro line. Li and Ni (2022) addresses the train scheduling problem for mainline railways by a multi-agent deep reinforcement learning method. A general environment that captures the system dynamics of the single-track and double-track railway systems is developed for DRL. Headway constraints and station capacities are included to ensure operational safety. Zhu et al. (2023) proposes a deep reinforcement learning framework for train delay management at a network-level for mainline railways, aiming to minimize passenger destination delays. The simulation model employed as the DRL environment includes train operations and passenger behaviors. The stations and line segments are also modeled as resource nodes. Yang et al. (2023) develops a deep reinforcement learning approach for the bi-direction single-track train scheduling problem. A two-dimensional time-space-capacity gridworld is designed to be the environment of DRL. It can record the dynamic states of the time and space dimensions in the train operations.

Currently, to the best of our knowledge, DRL methods have not been applied to TPRP. The above reviewed studies regarding train scheduling all focus on the macroscopic train scheduling problem on the railway lines. An essential reason is that, the environment of DRL, describing train operation dynamics with running times and capacity occupations at the macroscopic level, is easy to implement by numerical simulations and simplifying railway stations to resource nodes with certain capacities. A sophisticated simulation technique is not necessary. On the contrary, thanks to leveraging the discrete-event simulation model of train operations in a station as the environment, our L2RP method offers a fresh perspective on applying DRL methods to solve train scheduling problems at the microscopic level.

3. Problem description

In this section, we first elaborate on the detailed settings of the TPRP in this paper, providing a clear study scope. Referring to our previous work (Lu et al., 2022), we then briefly introduce the concept of Degree of Conflict (DOC) and the constraints of TPRP. DOC is the basis of modeling route conflicts based on sections. All notations in this section are listed in Table A.1 in Appendix A.

3.1. Problem settings

This paper focuses on the train platforming and rescheduling problem, which is an operational-level extension of the typical train platforming problem. Given a timetable including a set of trains (denoted as set $\mathcal{H}$) to be operated in the railway station, the timetable specifies the scheduled arrival time t_h^a and scheduled departure time t_h^d for each train $h \in \mathcal{H}$, while a corresponding platforming plan designates the scheduled platform b_h for each train h. This timetable and platforming plan are feasible if there are no train delays, since they are determined at the tactical level and validated before being put into practical operation. However, if trains cannot arrive at the station as scheduled due to disturbances, they must be 'rescheduled' based on safety requirements to operate with new time slots, platforms, and routes, aiming to ensure conflict-free train operations and minimize train delays. This is the train platforming and rescheduling problem studied in this paper.

In detail, given the primary delay of specific trains in $\mathcal{H}$, TPRP opts to re-order trains and re-assign platforms to minimize train knock-on delays. In real-time train operations, because of operational disturbances (temporary speed limits due to bad weather, operation interruptions due to track failures, etc), trains may have delays when arriving at the station, called *primary delays* (Yuan and Hansen, 2007). Primary delays lead to train operation deviations from the original timetable. Such deviations bring route conflicts and platform contentions among trains, because the original scheduled receiving route, departure route, and berthing track may be occupied by other trains when the delayed trains arrive, which results in infeasibility to the original timetable and platforming plan. Therefore, each train h with primary delay Δ_h has to be rescheduled for assigning conflict-free routes and an idle platform, ultimately attaining a new feasible timetable and platforming plan for all trains. Trains without primary delays in the timetable may also be adjusted to use alternative routes and platforms to match the rescheduling for delayed trains. If train h's actual arrival/departure time differs from the scheduled time, a *knock-on delay* appears. To minimize the total knock-on delays of trains and recover train operations to the original schedule, a new feasible timetable and platforming plan are expected to be provided for all trains in real-time by re-ordering arrivals, re-assigning platforms, and re-ordering departures simultaneously.

The route and platform information of the railway station is a given input. Considering a railway station with a set of berthing tracks, denoted as set $\mathcal{B}$. An example is Fig. 1, where there are six berthing tracks. The entry/exit points are denoted as $u \in \mathcal{U}$ and $v \in \mathcal{V}$, respectively. The route in the railway station is denoted as set $\mathcal{R}$, including receiving routes $\mathcal{R}_p \subset \mathcal{R}$ and departure routes $\mathcal{R}_q \subset \mathcal{R}$. The operation process of a train in the railway station is that it arrives at the station from the entry point via a receiving route, dwells on the berthing track, and leaves the station via a departure route and the exit point. Between an entry/exit point and a berthing track, it is possible that more than one receiving/departure route is physically available. Here, we assume that only one receiving/departure route alternative links a boundary point and a berthing track, considering we focus on the platforming problem rather than the routing problem. This is a widely used setting, such as (Sels et al., 2014; Lu et al., 2022; Zhang et al., 2025b). Under this

setting, when rescheduling a train, its optional receiving routes, departure routes, and berthing tracks can be determined according to its entry and exit points. Given a train h entering the station from entry point $u_h \in \mathcal{U}$ and leaving the station through exit point $v_h \in \mathcal{V}$, the receiving routes connecting u_h and departure routes connecting v_h are

$$\begin{aligned} \mathcal{R}^{p,u_h} &= \{r | u_r = u_h, \forall r \in \mathcal{R}^p\} \\ \mathcal{R}^{p,v_h} &= \{r | v_r = v_h, \forall r \in \mathcal{R}^q\} \end{aligned} \tag{1}$$

where u_r is the entry point of receiving route r and v_r is the exit point of departure route r. If a berthing track is simultaneously connected to u_h and v_h by routes, it can be assigned to train h. Hence, the optional platforms for train h can be determined as

$$\mathcal{B}_h = \{b^r | \forall r \in \mathcal{R}^{p,u_h}\} \cap \{b^r | \forall r \in \mathcal{R}^{p,v_h}\} \tag{2}$$

where b^r is the berthing track connected to route r. The optional receiving and departure routes for train h, denoted as $\mathcal{R}_h^p$ and $\mathcal{R}_h^q$ respectively, can be finalized according to the condition that a route connects an optional platform and the entry/exit point, as below:

$$\begin{aligned} \mathcal{R}_h^p &= \{r | b^r \in \mathcal{B}_h, \forall r \in \mathcal{R}^{p,u_h}\} \\ \mathcal{R}_h^q &= \{r | b^r \in \mathcal{B}_h, \forall r \in \mathcal{R}^{q,v_h}\} \end{aligned} \tag{3}$$

The original timetable, the original platform plan, the train primary delays, and the above optional routes and platforms for each train are the inputs for the TRPP. The solution of TPRP specifies optimized routes, platforms, and time slots for trains, including (1) the actual assigned platform β_h for train h (2) the actual used receiving route r_h^p and departure route r_h^q for train h (3) the actual arrival time τ_h^a, i.e., the time when train h stops at the assigned berthing track (4) the actual departure time τ_h^d, i.e., the time when train h departs from the berthing track (5) the actual receiving time τ_h^e, i.e., the time when train h enters the station at the entry point (6) the actual leaving time τ_h^z, i.e., the time when train h leaves the station at the exit point.

To provide high-precision solutions of TPRP, train operation processes and route conflicts in a station are required to be modeled at the microscopic spatial and temporal level. To this end, the RLSR interlocking mechanism is applied to ensure conflict-free routes, as we stated in Section 2.1. It enables us to model the track resources according to sections, which is a microscopic spatial perspective. Meanwhile, from the microscopic temporal perspective, a refined time unit (e.g., 1 second) in counting the occupation time of sections, routes, and platforms is able to guarantee the precision of the train operation processes, improving the potential of applications in real-world train delay situations. Considering that such requirements of refined modeling usually lead to high computational costs, we use essential pre-calculated data to facilitate modeling train operation processes and route conflicts, which are introduced in the next section.

3.2. Degree of conflict

In this paper, train operations are modeled using the RLSR interlocking mechanism. To model the route conflicts based on sections at the microscopic level, we recall previous work (Lu et al., 2022) to illustrate a significant definition: Degree of Conflict (DOC). The DOC data is defined for route pairs. Consider an ordered conflict route pair r_1 and r_2, the degree of conflict of this pair, denoted as γ_{r_1,r_2}, is the shortest duration within which the access must be denied to any train using route r_2, after route r_1 is claimed for a train. For example, for route r_1 and r_2 in Fig. 1, the value of γ_{r_1,r_2} is the running time from the head of train 1 arrives at entry point u_1 to the tail of train 1 clears section ϵ_4, and adding a safety margin κ. The DOC value of a route with itself is the route travel time adding the safety margin. For two parallel routes, i.e., any two routes without shared sections, the value of DOC of them is 0. We note that DOC is essentially a separation time between two consecutive trains using a same facility, which is a widely applied idea in previous studies (Corman et al., 2009; Pellegrini et al., 2014; Sels et al., 2014). More details about DOC can be found in Lu et al. (2022). In this paper, the DOC data is pre-calculated based on the route-lock-section-release interlocking mechanism. A tiny spatial step based train kinematics simulation (see (Zhang et al., 2024; Lu et al., 2025)) is applied to complete the computation. In the computation settings, we assume that all trains have the same length, which is the longest length of possible rolling stocks, guaranteeing that shorter trains only bring larger safety margins.

The DOC data specifies the minimal temporal separation for two trains to use any two routes, and the temporal separation is determined by the spatial resources (i.e., the shared sections). Hence, the DOC data ensures the microscopic modeling of train operation processes and route conflicts. Using this pre-calculated data, when rescheduling trains, route conflicts can be identified immediately without real-time computation, extremely alleviating the computational pressure of solving the TPRP. In Section 4, we will elaborate on using DOC data to establish the Markov decision process and the corresponding state transitions.

3.3. The optimization objective and constraints of TPRP

We also recall the mathematical model in previous work (Lu et al., 2022) to introduce the optimization objective and constraints of TPRP. The purpose of our work is to minimize the total knock-on delays and the number of reassigned platforms simultaneously for all trains. The objective function is computed using the total deviation between the scheduled and actual arrival time, departure time, and platform reassignment for all trains, i.e.,

$$min. \quad obj = \sum_{h \in \mathcal{H}} (\tau_h^a - t_h^a) + \sum_{h \in \mathcal{H}} (\tau_h^d - t_h^d) - \sum_{h \in \mathcal{H}} x_{b_h,\beta_h} \tag{4}$$

where b_h is the assigned berthing track for train h in the original timetable, β_h is the actual used berthing track in the finalized solution. If b_h and β_h are the same, then $x_{b_h,\beta_h} = 1$; Otherwise, train h has been dispatched to a new platform, and $x_{b_h,\beta_h} = 0$. The time deviation is calculated in seconds.

In this objective function, one platform reassignment is explicitly defined as equivalent to one second of train delay. Such weighting reflects the practical operation priorities. From the perspective of train dispatchers, reassigning a platform is considerably 'cheaper' than allowing additional delays to accumulate. Unresolved train delays can propagate along the mainlines rapidly, affecting subsequent trains and leading to widespread disruptions. Furthermore, even from the passenger perspective, train operation punctuality is generally more critical than which platform they board/alight trains (Luangboriboon et al., 2025). Therefore, by setting a relatively low cost for platform reassignment and counting delays in seconds, this objective encourages decisions to prioritize the significant delay reduction over strict adherence to original platform assignments. This practical design indicates that the primary purpose is to minimize total knock-on delays, while the secondary purpose is to maintain preferred platform assignment if possible.

In addition, both arrival and departure delays are considered equally important in this objective. Because, on the one hand, these two metrics together provide a complete picture of the level of schedule adherence in the station; on the other hand, they reflect the congestion in train arrival and departure, respectively, and have equal influence on the availability of platforms and routes. Also, for a single train, its arrival and departure delay could be different; for instance, a train arrives on time but its departure is delayed. Here, we need to emphasize that only knock-on delays are computed in the objective. If a train's arrival is delayed, the primary delay will be added to update its scheduled time to ensure the objective still only includes knock-on delays.

The constraints of TPRP contain three aspects. First, the train operational constraints require a train not to arrive before its scheduled arrival time and not to depart before its scheduled departure time. Early arrival is not allowed since it would not be feasible due to the speed limitations and headway requirements on the mainline. Early departure is not allowed since it could exceed capacity on the mainline or incur complaints from last-minute boarding passengers. It should be noted that if a primary delay occurs to a train, its scheduled arrival time is updated by adding the primary delay time, and its scheduled departure time is then updated using the larger one between the updated arrival time plus the minimum dwell time and the original departure time. Such an update allows a train arriving late to dwell only for the minimum dwell time but also forbids it from departing before the original schedule. Second, the platform conflict avoidance constraints require each train to select only one platform and the occupation of a platform by two trains has to be separated by a clearance buffer. Note that the time between two occupation on a berthing track is calculated by a train's actual arrival time and its predecessor's actual departure time. Last, the route conflict avoidance constraints require the time interval between the claims of two conflicting routes to be larger than the pre-calculated DOC value. For the mathematical formulation of the above constraints, which is not the emphasis of this study, we refer interested readers to previous work (Lu et al., 2022). Next, we turn to details of the proposed deep reinforcement learning method for the TPRP.

4. Methodology

This section elaborates on the proposed L2RP method for the TPRP. To enhance accessibility for a broader audience, we first provide an intuitive overview of our DRL based method before delving into technical details. In simple terms, the DRL agent of our method acts as an intelligent decision generator that can make platforming and rescheduling decisions in a simulated environment. In each decision cycle, the environment provides a snapshot of the current station situation, including the status of trains, platforms, and routes. The DRL agent analyzes the observed situation, evaluates possible platforming and scheduling actions, and then selects one, for example, assigning an inbound train to a platform. Once the decision is applied, the environment updates the station status accordingly, and gives feedback (e.g., train delay time). By repeating this decision-feedback iteration and learning from the feedback continuously, the DRL agent is expected to gradually learn effective platforming and rescheduling strategies that can minimize knock-on delays under different scenarios.

Technically, the above iteration process is illustrated in Fig. 2. A microscopic discrete-event simulation model serves as the interaction environment for the DRL agent. It implements the formulated MDP model of solving TPRP. It provides the current state, executes actions, calculates rewards, and transitions to the next state. The state, containing the status of inbound trains, routes, and berthing tracks, is encoded into a graph structure to represent the interdependencies among them. Then, a hybrid graph neural network (hAI-GNN) 'analyzes' the graph to learn informative features (i.e., node embeddings), which are passed to the action reasoning network (ARN) to 'evaluate' actions. The selected action is then executed in the simulation model to drive the transition and yield a reward, completing an iteration step. The policy network (i.e., constituted by hAI-GNN and ARN) governing the DRL agent is trained by the proximal policy optimization algorithm. The details of MDP modeling, discrete-event simulation, policy network design, and training procedure are introduced in the following subsections.

4.1. MDP model

The solving process of the TPRP in this study is modeled as a MDP, including state, action, transition, and reward. The MDP model is implemented by interacting with an environment. All notations related to the MDP model are also listed in Table A.1 in Appendix A.

4.1.1. State

To make intelligent platforming and rescheduling decisions, the DRL agent requires the state to capture the realistic train operational situation in a quantitative way. Specifically, a state s_i, located at the i-th step of a decision episode, reflects the status of

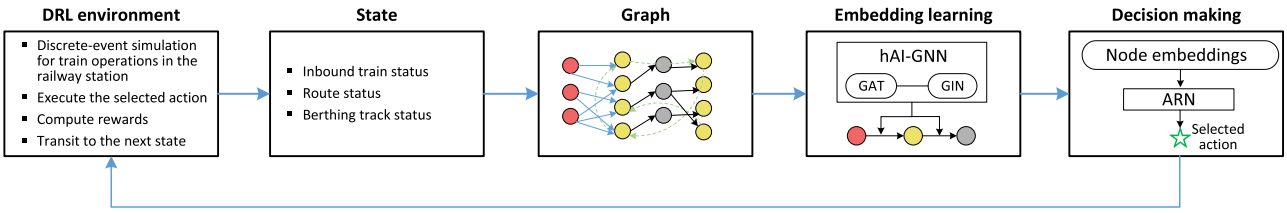


Fig. 2. The framework of our L2RP method.

Table 2
The features of inbound trains, berthing tracks, and routes included in a state.

Item	Features (Static and dynamic feature are start with ∗ and •, respectively)
Inbound train	∗ Scheduled arrival time • Binary label of decision object train • Postponing status • Inbound train operation stage (not reach the earliest receiving time; postponed to arrive; in the arrival running process; dwell at the assigned platform) • Actual receiving time • Actual arrival time ∗ Scheduled platform index • Assigned platform index ∗ Scheduled departure time • Current system time
Berthing track	∗ Platform index • Current occupation status • Actual arrival time of the current outbound train • Outbound train operation stage (Not reach the scheduled departure time; postponed to depart; in the departure running process) • Binary label of decision object train • Scheduled departure time of the current outbound train • Postponing status of the current outbound train • Actual departure time of the current outbound train • Next available time of the berthing track • Current system time
Route	∗ Route type (receiving/departure) ∗ Route travel time • Last claiming time • Next available time • Current availability status • Current system time

inbound trains, routes, and berthing tracks in the railway station, which are essential considerations for decision-making in train rescheduling and platform assignments. It should be noted that inbound trains are described in the state directly, while outbound trains are reflected by the status of berthing tracks, because only trains that have already stopped on berthing tracks can be outbound trains. The specific included information in a state is listed in Table 2. We choose several straightforward but crucial features for inbound trains, berthing tracks, and routes, respectively. Static and dynamic status of them are both considered. For example, the scheduled arrival/departure time of trains, the route type, and the train traversal time on a route are fixed features; the train operation stage, the actual arrival/departure time of trains, and the next available time of routes are dynamic features, varying under different states according to train operations in the railway station.

These features describe the status of train operations and utilization of routes and berthing tracks, which are easy to obtain without a heavy manual workload of specialized tuning or computation. However, such simple features may not be sufficient to achieve reasonable decision-making, because they do not directly reflect the complex interdependency existing among trains, routes, and berthing tracks. For example, the optional receiving routes, berthing tracks, and departure routes of a train are different according to its entry and exit point (see Eqs. (1) to (3) in Section 3.1), and conflicts may arise among multiple routes during train operation processes (see Section 2.1). Therefore, in this paper, to maintain a light workload in crafting features and simultaneously guarantee sufficient and effective information for train platforming and rescheduling decision-making, we encode a state to a graph structure and leverage Graph Neural Networks (GNN) to learn informative state representations. In other words, under a state, we only provide very raw features and leave the task of extracting implicit but effective information to the GNN. The graph structure and GNN are detailed in Section 4.2 and Section 4.3.1, respectively.

Notably, in Table 2, we set a feature of whether an inbound/outbound train is the decision object train. In our MDP model, a state s_i has a unique *decision object train* h_i. Hence, this feature is used to directly designate which train is the decision object train

under the current state. In the following Section 4.1.3, we will elaborate on how to determine a state and a decision object train synchronously.

4.1.2. Action

Under state s_i, the DRL agent must select an action a_i, altering the train operations in the station. In the context of solving TPRP, the most fundamental decisions involve assigning trains to platforms and determining the sequence of train movements. Hence, following this essential idea, the actions are designed around platform assignment and train sequencing, driving the DRL agent to learn effective strategies on train platforming and rescheduling. Specifically, two types of actions are designed as follows.

Dispatching action. Considering that train arrival and departure processes require definite specified routes, in our MDP model, we define the ***dispatching action*** a^r, which means selecting route r for a train, and the train uses the route to arrive or depart. Hence, the set of all possible dispatching actions is $\{a^r | r \in \mathcal{R}\}$. Under state s_i, decision object train h_i is given, if the selected action a_i is a dispatching action, i.e., $a_i \in \{a^r | r \in \mathcal{R}\}$, the corresponding decision is train h_i to run via route r immediately. Hence, the dispatching action mainly focuses on making assignment decisions and triggering train operations.

It should be emphasized that the function of a dispatching action a_r covers the decision-making requirements of platform assignment. Given the fact that only one berthing track is connected to a receiving route, selecting a receiving route $r \in \mathcal{R}^p$ for an inbound train equals assigning the connected platform to the train. In such a case, a dispatching action a^r means dispatching the inbound train h_i to arrive via receiving route r immediately under state s_i, and assigning the platform connected to route r (i.e., b^r) for the inbound train to dwell.

When applying to an outbound train, a dispatching action is only used to trigger the train to depart, without the function of platform assignment, because an outbound train h_i has already been dwelling on the berthing track. Hence, in this situation, dispatching action a^r means dispatching the outbound train h_i to depart via departure route $r \in \mathcal{R}^q$ immediately under state s_i. We also note that, as we mentioned in Section 3.1, only one departure route connects the berthing track and the outbound train's specified exit point, which is the only route that can be scheduled for the outbound train.

Postponing action. To augment the DRL agent with the rescheduling capability, we design a ***postponing action*** a^ω. In the process of solving TPRP, trains may be required to arrive or depart at a later time, which is forced due to no available routes under the current state or is proactive to operate other trains first for the ultimate purpose of minimizing total knock-on delays. To this end, a postponing action a^ω is added to reschedule train arrival and departure. If the postponing action is decided for decision object train h_i, i.e., $a_i = a^\omega$, then no routes and berthing tracks are allocated to train h_i for the time being. Train h_i will be held to wait for the succeeding decision moments. Once the postponed action is executed for a train, the train will have knock-on delays. Such trains are called *postponed trains* to be distinguished from trains with only primary delays.

It is evident that, the total number of all actions equals the number of routes adding the one postponing action, i.e., $|\mathcal{R}| + 1$. In other words, the size of this 1-dimensional action space is only determined by the number of routes in the railway station, instead of establishing a 2-dimensional mapping between all trains in the timetable and platforms in the railway station. In doing so, such a lightweight action space enables the DRL agent to focus on learning effective policies of assigning berthing tracks and scheduling routes for trains. Meanwhile, if multiple routes between an entry/exit point and a platform have to be considered, then including the associated dispatching actions representing these routes to expand the action space can perfectly cover this requirement without extra train-related methodological implementation. Since no train information is included in the action space, a resulting subsequent task is to determine the decision object train associated with each state, which is introduced in the following sections.

4.1.3. Integrated transition and environment

Under a state, executing a decision (i.e., dispatching action or postponing action introduced above) will lead to specific train operations, thereby advancing the state to the next. Such a process is called transition. It is required to specify how the selected action moves trains, occupies routes, and potentially causes new delays during a transition, obtaining the next state. To comprehensively model this transition process, in our L2RP method, a discrete-event simulation model of train operations in the railway station is developed. The DRL agent observes the simulation model to collect information for a state, while the train operations specified by the action are implemented in the simulation model, thus the DRL agent is aware of the changed state.

Fig. 3 is an illustrative diagram of a transition step. Generally, state s_i is observed by the DRL agent from trains, routes, and berthing tracks in the simulation model. A masked action space A_i associated with decision object train h_i will also be provided to the DRL agent, since route availability varies according to train operations and route conflicts. The action a_i by the DRL agent is then executed in the simulation model. It means the train operations specified by action a_i are implemented, and the status of trains, routes, and berthing tracks is accordingly updated. Synchronously, tuples of succeeding decision moment and decision object train are generated and used to determine the next state. Then, the state can transit from the current to the next. Such a transition step will be iteratively implemented until the terminal state (e.g., the last train departs from the station in our TPRP), then an entire decision episode is completed.

In the following part of this section, we first elaborate on how to determine the succeeding decision moments and decision object train under the current state. Then, we introduce the DRL learning environment, which is a discrete-event simulation model specialized for the state transitions.

Succeeding decision moment & decision object train. In our MDP model, a state is established at a decision moment with the unique decision object train. For a state s_i, we denote the decision moments and decision object train as a tuple (t_i, h_i). To accomplish the state transition from s_i to s_{i+1}, according to the status of trains, routes, and berthing tracks updated by the selected action a_i, we generate the tuples of succeeding decision moment and decision object train under state s_i, which is denoted as set M_i. A tuple means

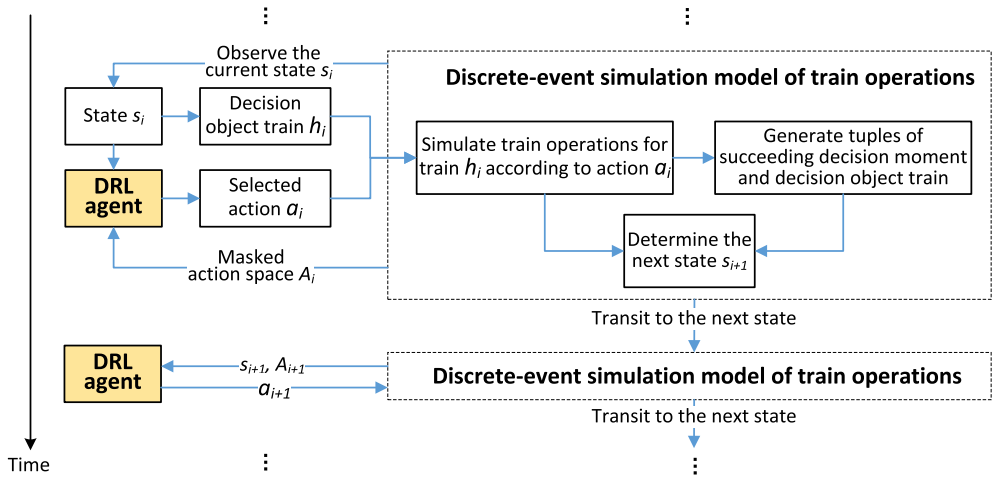


Fig. 3. An illustration of transitions and environment in our L2RP method.

a possible decision moment for the DRL agent to make a decision for the corresponding decision object train. The next state s_{i+1} will be established by the tuple of the earliest decision moment in M_i. Specifically, the set of tuples of succeeding decision moments and decision object train under state s_i (i.e., M_i) contains 4 parts, which are determined according to the following 4 different situations, respectively.

- *Earliest receiving time.* Considering that scheduled arrival time t_h^a specifies the time of a train stops at the berthing track, we define the train earliest receiving time t_h^e. It is the time of train h entering the station at the entry point and can be calculated as

$$t_h^e = \min_{r \in \mathcal{R}_h^q} t_h^a - y_r \tag{5}$$

where y_r is the train travel time on route r. If train h arrives via receiving route r, it should enter the station at time $t_h^a - y_r$, i.e., scheduled receiving time via route r, since trains are required not to arrive before the scheduled arrival time. To guarantee train h can arrive at the berthing track on time regardless of which receiving route is selected, we use the minimal scheduled receiving time to define its earliest receiving time t_h^e. A state set by (t_h^e, h) will be the first state to make a decision for inbound train h. Under state s_i, the tuples of succeeding decision moment and decision object train, which are set using the earliest receiving time of trains, denoted as set M_i^1, can be expressed as

$$M_i^1 = \{(t_h^e, h) | t_h^e > t_i, \forall h \in \mathcal{H}_{s_i}^{in}\} \tag{6}$$

where $\mathcal{H}_{s_i}^{in}$ contains the considered inbound trains under state s_i, t_i is the current decision moment of state s_i. The tuples with inbound trains that have not reached their earliest receiving time currently are included in M_i^1.
- *Updated scheduled departure time.* If train h arrives at the berthing track punctually, its scheduled departure time will not change. On the contrary, if train h is postponed to arrive, its scheduled departure time should be updated according to its actual arrival time and the specified dwell time. Therefore, the updated train scheduled departure time can be calculated as

$$\hat{t}_h^d = \tau_h^a + (t_h^d - t_h^a) \tag{7}$$

where τ_h^a is the actual arrival time of train h. That is, after a dispatching action a^r associated with a receiving route $r \in \mathcal{R}^p$ is executed for train h, its actual arrival time and updated scheduled departure time are determined. Correspondingly, a state set by $(\hat{t}_h^d, h)$ is a state to attempt train h's departure. Under state s_i, the tuples of succeeding decision moment and decision object train which are set by updated scheduled departure time of trains, denoted as set M_i^2, can be expressed as

$$M_i^2 = \{(\hat{t}_h^d, h) | \hat{t}_h^d > t_i, \forall h \in \mathcal{H}_{s_i}^{out}\} \tag{8}$$

where set $\mathcal{H}_{s_i}^{out}$ contains the outbound trains currently dwelling on the berthing tracks under state s_i, and tuples with outbound trains that have not reach the updated scheduled departure time are included in M_i^2.
- *Waiting time limit.* After a postponing action is selected, a train will be postponed to arrive or depart. To guarantee all postponed trains can be decided and avoid the DRL agent selecting the postponing action multiple times for a train, we set a waiting time limit as a decision moment for postponed trains. This decision moment can be calculated as

$$t_h^m = \tau_h^{a^\omega} + \varphi, \tag{9}$$

where t_h^m is the time that train h reaches the waiting time limit, $\tau_h^{a^\omega}$ is the time of selecting the previous postponing action for train h, and φ is the waiting time limit. Note that the waiting time limits are counted for the arrival and departure of a single

train independently. Under state s_i, the tuples of succeeding decision moment and decision object train which are set by waiting time limits, denoted as set M_i^3, can be expressed as

$$M_i^3 = \{(t_h^m, h) | t_h^m > t_i, \forall h \in \mathcal{H}_{s_i}^{wait}\} \tag{10}$$

where $\mathcal{H}_{s_i}^{wait}$ is the set of postponed trains which are waiting to arrival or departure under state s_i. The succeeding tuples with decision moments later than the current time t_i are included in M_i^3.

- *Next available time of routes.* Besides the waiting time limit, for postponed trains, we set tuples of succeeding decision moment and decision object train according to route availability, providing more chances for the DRL agent to make decisions for these trains. In our MDP, we model route availability by defining the ***next available time*** of a route, which is the earliest time that the route can be claimed for a train without route conflicts. As we stated in Section 3.2, the DOC data enables us to be aware of conflicting information between any two routes. Hence, when a dispatching action a^r is executed under a state, route r is claimed for a train, and then we can calculate the next available time of route r' that conflicts with route r as

$$\lambda_{r'} = \tau_{a^r} + \gamma_{r,r'}, \quad \forall r' \in \{r' \in \mathcal{R} | \gamma_{r,r'} > 0\} \tag{11}$$

where τ_{a^r} is the last claim time of route r by dispatching action a_r. $\lambda_{r'}$ the next available time of route r'. It is noted that, a departure route can be claimed for an outbound train directly if the route is available. However, to claim a receiving route for an inbound train, there is an extra condition that the berthing track connected to the receiving route should also be available. Hence, we calculate the next available time of berthing tracks and update the next available time of conflicting receiving routes r' as

$$\begin{aligned} &\lambda_b = \tau_{\tilde{h}}^d + \eta, \quad \forall b \in \mathcal{B} \\ &\lambda_{r'} = \max\{\lambda_{r'}, \lambda_b\}, \quad \forall r' \in \{r' \in \mathcal{R}^p | \gamma_{r,r'} > 0, b_{r'} = b, b \in \mathcal{B}\} \end{aligned} \tag{12}$$

where λ_b is the next available time of berthing track b, $\tilde{h}$ is the previous train assigned to berthing track b, $\tau_{\tilde{h}}^d$ is the actual departure time of train $\tilde{h}$, η is the safety buffer to separate two consecutive trains assigned to the same platform, $b_{r'}$ is the berthing track connected to receiving route r'. For each receiving route connecting berthing track b, its next available time will be updated if the next available time of berthing track b is larger.

Using Eqs. (11) and (12), the next available time of routes will be updated if selected action a_i is a dispatching action under state s_i. If a receiving(departure) route r is one of the optional receiving(departure) routes of a postponed train h waiting to arrival(departure), it means the DRL agent may dispatch train h to use route r in subsequent states. In this case, a tuple of succeeding decision moment and decision object train is set using the next available time of route r and train h. Therefore, under state s_i, set M_i^4 contains all tuples where the next available times of routes are later than the current time. It can be expressed as

$$M_i^4 = \{(\lambda_r, h) | \lambda_r > t_i, \forall h \in \mathcal{H}_{s_i}^{wait}, \forall r \in \mathcal{R}_h^p \cup \mathcal{R}_h^q\} \tag{13}$$

where $\mathcal{H}_{s_i}^{wait}$ is defined in Eq. (10), R_h^p and R_h^q are optional receiving and departure routes of train h, respectively.

At this point, under state s_i, we generate all tuples of succeeding decision moment and decision object train using the above 4 parts, i.e., set M_i. The next state s_{i+1} is then determined using the tuple in M_i with the earliest decision moments, which can be expressed as

$$\begin{aligned} &M_i = \bigcup_{j=1}^{4} M_i^j \\ &(t_{i+1}, h_{i+1}) = \underset{(t,h) \in M_i}{\operatorname{argmin}} (t, t_h^a) \end{aligned} \tag{14}$$

where the $\operatorname{argmin}(t, t_h^a)$ means we determine the tuple for the next state from the set M_i based on a temporal ascending order with the decision moment as the primary key and the scheduled arrival time of decision object train as the secondary key. The secondary key is set to give a clear order for tuples with the same decision moment. For example, tuples in M_i^4 may have the same decision moments, since probably more than one postponed train can use the same route. In this case, multiple tuples are set in M_i, where the same next available time of route is used as the decision moments, and the postponed trains are used as the decision object train.

Now, we have specified the next state s_{i+1} with its decision object train h_{i+1} synchronously, and the state can transit from s_i to s_{i+1}. In a decision episode, the first state s_1 is located at the earliest receiving time of the earliest train in the given timetable. Then, using this transition mechanism, the state transits iteratively until all trains planned in the given timetable have finished their arrival and departure processes, where a decision episode is completed.

The above transition mechanism ensures the DRL agent makes a decision only for the decision object train under a state. Hence, the lightweight action space introduced in Section 4.1.2, which focuses on dispatching a single train, aligns perfectly with this mechanism that operates the unique train per state. However, it is obvious that not all routes are available for a decision object train. For example, receiving routes can not be scheduled for an outbound train. Therefore, to avoid meaningless exploration and improve training efficiency, an action mask is applied to refine the specific action space under a state. The generation principle of action masks relies on the decision moment categorizing, which can be summarized as: If the decision object train is triggered by *the next available time of a route*, then the masked action space contains the unique dispatch action associated with this route; otherwise, the masked action space will contain dispatch actions associated with all current available routes. Meanwhile, the postponing action will

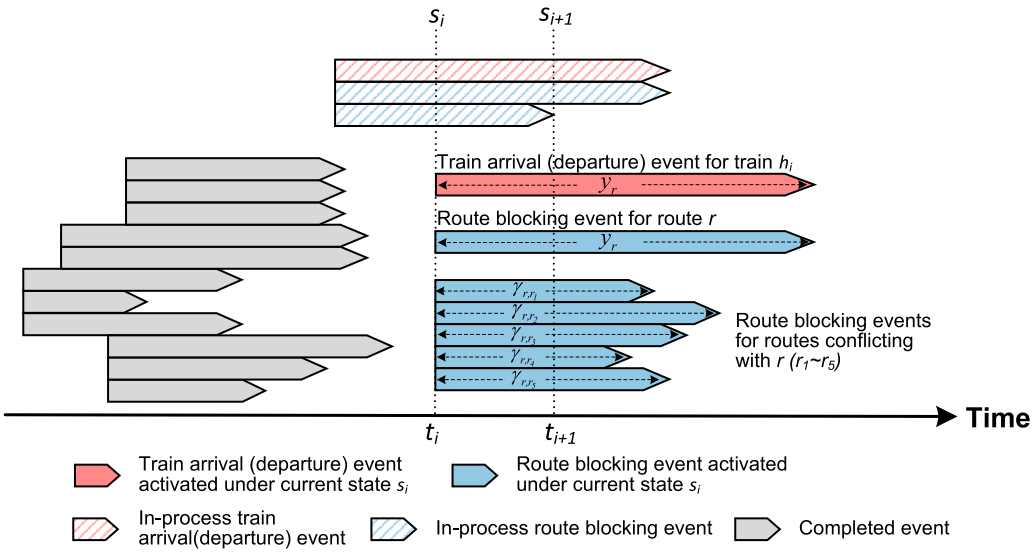


Fig. 4. An illustration of activated events under current state in the simulation model.

be added to the masked action space if the decision object train has not exceeded the waiting time limit. The formula description of the action mask is provided in Appendix B for interested readers.

Environment. As we stated before, in our L2RP method, a discrete-event microscopic simulation model of train operations in the railway station is employed as the environment. This environment is used to observe a state, generate the masked action space, and execute the selected action by the DRL agent, as we stated in Fig. 3. Meanwhile, to collaborate with the transition mechanism introduced above, the simulation model is advanced according to the specified next state after an action is executed.

In specific, three types of events to model train operations are designed in the simulation model, including train arrival event, train departure event, and route blocking event. These events are activated when a dispatching action is selected as a_i under the current state s_i, and the status of trains, routes, and berthing tracks are updated accordingly. As shown in Fig. 4, a train arrival event is activated for train h_i if a dispatching action is associated with a receiving route r. Similarly, a train departure event is activated if the dispatching action specifies a departure route r. The duration of train arrival and departure events equals the train travel time on the assigned route (i.e., y_r). The train operation stages and occupation status of berthing tracks are also updated according to the train arrival and departure events. Meanwhile, a route blocking event with the duration of y_r is activated to model the route occupation on route r. Furthermore, more route blocking events are activated to model the route conflicts. As the example of route $r_1 \sim r_5$ in Fig. 4 , the duration of these route blocking events are specified using the DOC data directly, instead of imitating the occupation and releasing processes of sections of routes. In doing so, we are able to not only accelerate the simulation speed, but also quickly obtain the route next available time required to generate succeeding decision moments. Additionally, if the postponing action is selected as a_i, no events will be activated, but a waiting time limit will be set in the status of train h_i.

Referring to Eqs. (5) to (14), the tuples of succeeding decision moments and decision object trains under state s_i is generated according to the activated events and status of trains and routes in the simulation model. Then, the next state s_{i+1} can be determined by the transition mechanism, and the simulation model is advanced to the decision moment of state s_{i+1}. All activated events are advanced in parallel during this process, representing train running processes and route occupation and releasing with time elapsing.

We note that the above event-triggered transition mechanism and the coordinated discrete-event simulation model do not employ any time steps to construct train operations and decision episodes. The decision moments and event duration are determined according to train receiving/departure times, route travel times, and DOC data. Therefore, the time precision in our MDP is identical to the provided input data, e.g., 1 sec in the paper. Moreover, the DOC data used in this paper is calculated under the route-lock-section-release interlocking mechanism. Hence, the simulation model is developed at the microscopic level, where the spatial unit are sections of tracks and switches and the temporal unit is 1 sec. Integrating such a high-fidelity simulation modeling into the DRL framework ensures our L2RP method can deliver high-precision solutions for the TPRP, consequently enhancing the feasibility of applying the solutions in real-world delay scenarios.

4.1.4. Reward

The reward function is the primary mechanism for guiding the DRL agent towards the goal of TPRP. A reward must be provided after the selected action is executed in the simulation model. It aims to inform the DRL agent quantitatively about whether the selected action positively or negatively influences the train delays in the station, encouraging the DRL agent to effectively learn decision-making strategies of train platforming and rescheduling. In our MDP model, the reward depends on the selected action under a state. Following the objective function as shown in Eq. (4), we design different rewards as Eq. (15) for the dispatching action and

postponing action, respectively, aiming to guide the learning process for minimizing the total knock-on delays and changed platforms.

$$\mu_i = \begin{cases} -((\tau^a_{h_i} - t^a_{h_i}) \times 2 + \sigma \times (1 - x_{b_{h_i},\beta_{h_i}})), & \text{a dispatching action with a receiving route for } h_i \quad (15a) \\ -((\tau^d_{h_i} - t^d_{h_i}) - (\tau^a_{h_i} - t^a_{h_i})), & \text{a dispatching action with a departure route for } h_i \quad (15b) \\ -2\varphi, & \text{a postponing action for train } h_i\text{'s arrival} \quad (15c) \\ 0, & \text{a postponing action for train } h_i\text{'s departure} \quad (15d) \end{cases} \tag{15}$$

In Eq. (15), μ_i is the reward of executing action a_i under state s_i; $x_{b_{h_i},\beta_{h_i}}$ is the binary flag for indicating whether the platform assigned to h_i has changed or not, which is defined in Eq. (4); φ is defined in Eq. (9). Eqs. (15a) and (15b) are the rewards for dispatching actions associated with receiving routes and departure routes, respectively, which are mainly determined by the values of knock-on delays. We note that if a train is postponed in the arrival process, its arrival knock-on delay will be counted again for calculating the departure knock-on delay. Hence, to accurately credit the contribution of actions, we double the value of the arrival knock-on delay in the reward of selecting a dispatching action with a receiving route, and deduct the arrival knock-on delay from the departure knock-on delay when a dispatching action with a departure route is selected for the same train. It should be emphasized that only the dispatching action associated with a receiving route can decide which berthing track is assigned to a train, so the penalty (i.e., σ) of reassigning a new berthing track to a train is added to the reward of such an action, as shown in Eq. (15)a. This penalty term is a reward-shaping technique designed to guide the DRL agent toward achieving the secondary objective, i.e., minimizing platform changes. Since the value of train knock-on delays in Eq. (15)a are counted in seconds, leading to values amounting to hundreds or thousands, the value of σ should be large enough to scale up the penalty for each platform change appropriately. This ensures that the DRL agent can sufficiently recognize the platform changes, rather than ignoring them due to the scale disparity. For the rewards of the postponing action, we leverage the value of the waiting time limit as the penalty. Similar to the dispatching action, the doubling and deducting design is also applied to the postponing action for arrival and departure, respectively, since the waiting time before arrival will be repeatedly counted when trains depart. We note that reinforcement learning aims to maximize the cumulative rewards, thus, all values in Eq. (15) are negative implying penalties. Maximizing cumulative rewards is achieving the purpose of minimizing the total knock-on delays and the number of reassigned berthing tracks.

4.2. Encoding state as graph

Until now, the MDP model and the environment of solving the TPRP have been established, except for the last piece of the puzzle: the detailed state representations. For the DRL agent to make reasonable platforming and rescheduling decisions, the state representation must not only cover individual features listed in Table 2 but also be able to capture complex interdependencies between them. Therefore, to naturally model the spatial and temporal relationships among trains, routes, and berthing tracks, and represent their dynamic and informative status, we encode the state as a graph structure.

Specifically, a graph G_i is designed to represent state s_i, i.e., $G_i = \{\mathcal{V}^i_H, \mathcal{V}_R, \mathcal{V}_B, \mathcal{L}^i_{as}, \mathcal{L}_{cn}, \mathcal{L}_{cc}\}$. An example is illustrated in Fig. 5. $\mathcal{V}^i_H$ is the set of inbound train nodes under state s_i. It should be emphasized that, under a state s_i, $\mathcal{V}^i_H$ only contains n inbound trains to be arranged in the future, not all inbound trains in the given timetable. n is a fixed and relatively small number, set to 20 in this paper. These trains (1) have not reached the earliest receiving time or (2) have reached the scheduled arrival time but are postponed to be assigned with a berthing track. $\mathcal{V}_R$ is the node set of all routes in the railway station, including the two independent subsets of receiving and departure routes ($\mathcal{V}^p_R$ and $\mathcal{V}^q_R$). $\mathcal{V}_B$ is the set of berthing track nodes, representing all berthing track nodes in the railway station. There are 3 types of edges linking nodes in the graph. The first type is the *assignment edge* (denoted by set $\mathcal{L}^i_{as}$), linking an inbound train node and a receiving route node. Given an inbound train h, the assignment edges are established between the node of train h and the nodes of its optional receiving routes (i.e., $\mathcal{R}^p_h$). Hence, the assignment edges can include berthing track assignment possibilities into the graph. The second type is the *connection edge* (denoted by set $\mathcal{L}_{cn}$), linking route nodes and berthing track nodes. For a receiving/departure route and the berthing track connected with it, a connection edge is added from the route node to the berthing track node, representing the physical accessibility of the railway station. The last type is the *conflict edge* (denoted by set $\mathcal{L}_{cc}$), linking the nodes of any two routes that could conflict with each other. Thanks to the pre-calculated DOC data, we can easily recognize route pairs with potential conflicts according to positive DOC values. We note that a DOC value γ_{r_1,r_2} is set for the ordered occupation of route r_1 and r_2, indicating γ_{r_1,r_2} may not equal γ_{r_2,r_1} due to the different order of occupation. Hence, we establish two directional edges in opposite directions between two route nodes with positive DOC values, which can intuitively represent the occupation order and conflict relation between any two routes. No conflict edge is built between nodes of two parallel routes, i.e., two routes without meaningful DOC values.

The graph encoded state is able to regulate the computational cost in solving iterations. The fixed number of inbound train nodes under a state enables the DRL agent to consider only inbound trains that will arrive in a small period in the future. The reason for this design is, according to the delay propagation process, the quality of scheduling decisions for current trains may have little influence to trains arriving much later in the future, especially if the given timetable has sufficient slack time. Hence, this design can exclude inappreciable information from later arriving trains for the current state, and help the DRL agent concentrate on scheduling upcoming trains and trains postponed to be received. For the same consideration, the included outbound trains in a state are limited to the ones that have been assigned to berthing tracks, represented by features of berthing track nodes in the graph encoded state. By these settings, the number of nodes and edges in the graph is foreseeable and reasonable, since the number of routes and berthing tracks in a railway station are both fixed and known. Benefiting from such a size-limited graph, the computation cost for the DRL agent to learn effective embeddings and select crucial actions based on states in the training and solving processes can be regulated at an

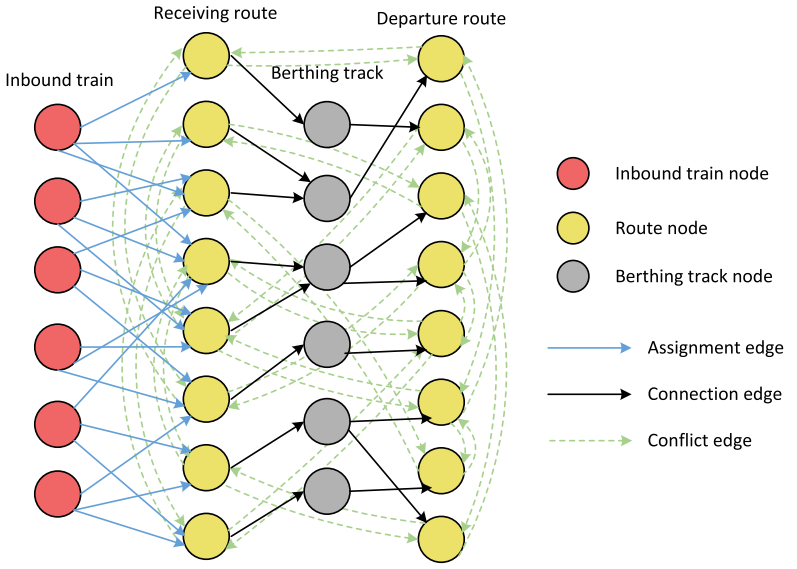


Fig. 5. The graph encoded state.

acceptable level. In addition, the size of the graph is only related to station layouts and independent of the number of trains in the timetable, enabling our L2RP method to deal with hundreds of trains with less computational pressure.

Based on the status and characteristics of trains, routes, and berthing tracks listed in Table 2, we define the raw feature vectors for nodes and edges in the graph. Given the node $h \in \mathcal{V}_H^i$, $r \in \mathcal{V}_R$, and $b \in \mathcal{V}_B$, their raw features are defined as $v_h \in \mathbb{R}^{10}$, $v_r \in \mathbb{R}^6$, and $v_b \in \mathbb{R}^{10}$, respectively. For an assignment edge $[h \Rightarrow r] \in \mathcal{L}_{as}^i$, its raw features are defined as $l_{as}^{h,r} \in \mathbb{R}^2$, including two binary flags that indicate whether the route r linked by the edge connects the scheduled platform of train h and whether the route r linked by the edge is selected for train h. For a conflict edge $[r_1 \Rightarrow r_2] \in \mathcal{L}_{cc}$, its raw features are defined as $l_{cc}^{r_1,r_2} \in \mathbb{R}^2$, including two values, which are the DOC value from route r_1 to route r_2 (i.e., γ_{r_1,r_2}) and the last time of claiming route r_1. No features are set for connection edges since they are only used to represent physical accessibility.

The graph structure and features of nodes and edges are updated along with state transitions to reflect train operation dynamics. For example, after a train arrival event ends in the simulation model, it means an inbound train clears the assigned receiving route and stops on the berthing track, the node representing this inbound train will be updated to represent a new inbound train, which is the one with the earliest scheduled arrival time among all inbound trains that have not been included in the graph. In addition, the node and edge features, such as actual receiving times, actual arrival times, actual departure times, route availability status, whether postponed, etc., are updated according to the train arrival and departure events in the simulation model. Since the above graph-encoded state and the MDP model are crucial to our L2RP method, we provide an illustrative example of them in Appendix C to further aid in understanding these core designs.

4.3. Policy network

The policy network is the decision-making brain of the DRL agent. Its role is to understand the graph encoded state and compute a probability distribution over all possible actions. To achieve this, the policy network consists of two parts. The first part seeks to analyze the state and extract meaningful knowledge based on the graph, and the second part aims to select an action based on the learned knowledge.

4.3.1. HAI-GNN for node embedding

Given a state s_i represented as graph G_i, an action is required to be selected according to information extracted from G_i. To this end, we propose a graph neural network to learn node embeddings over the graph encoded state. By the message-passing and aggregation processes in the graph neural network, the graph neural network can capture both local and global information for a node, enabling informative final node embeddings. Such embeddings are beneficial for making effective decisions in complex systems (e.g., train operations in a railway station). In this paper, to align with the characteristics of train operations, a hybrid architecture including Graph Attention Network (GAT) and Graph Isomorphism Network (GIN) is proposed to constitute the graph neural network, which is named hAI-GNN as shown in Fig. 6. The specific neural structure and the processes of messing-passing and aggregation in hAI-GNN are described as follows.

In a graph encoded state $G_i = \{\mathcal{V}_H^i, \mathcal{V}_R, \mathcal{V}_B, \mathcal{L}_{as}^i, \mathcal{L}_{cn}, \mathcal{L}_{cc}\}$, the message-passing procedure is (1) $\mathcal{V}_H^i \to \mathcal{V}_R$ (2) $\mathcal{V}_B \to \mathcal{V}_R$ (3) $\mathcal{V}_R \to \mathcal{V}_R$ (4) $\mathcal{V}_R \to \mathcal{V}_H^i$ (5) $\mathcal{V}_R \to \mathcal{V}_B$. The first two steps aim to load the information of inbound and outbound trains to route nodes, followed

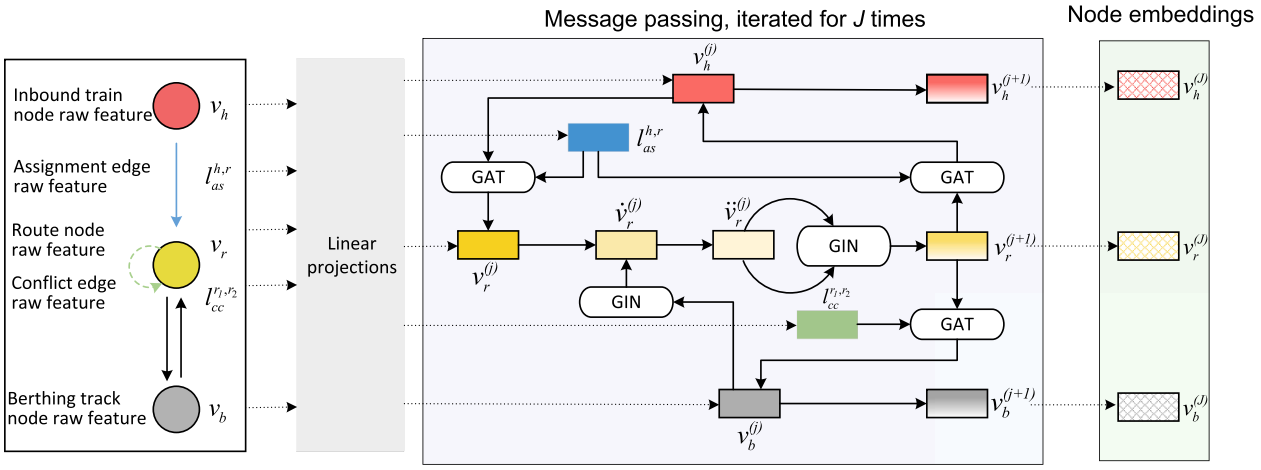


Fig. 6. The hAI-GNN for node embeddings.

by loading the current route status among route nodes themselves. Through these three steps, a route node embedding can gather information of trains, berthing tracks and other routes, providing a comprehensive evaluation of operation status for the dispatching actions associated with routes. The last two steps pass the information of route status to the inbound and outbound train, enabling a reasonable judgment of whether to apply the postponing action for an inbound/bound train. Before the message-passing among nodes, the raw features are first linearly projected to a uniform dimension as below:

$$\begin{aligned} &v_h^{(0)} = \mathbf{W_h^\rho} v_h, \mathbf{W_h^\rho} \in \mathbb{R}^{k\times 10}; \quad v_r^{(0)} = \mathbf{W_r^\rho} v_r, \mathbf{W_r^\rho} \in \mathbb{R}^{k\times 6}; \quad v_r^{(0)} = \mathbf{W_b^\rho} v_b, \mathbf{W_b^\rho} \in \mathbb{R}^{k\times 10}; \\ &l_{as}^{h,r} = \mathbf{W_{as}^\rho} l_{as}^{h,r}, \mathbf{W_{as}^\rho} \in \mathbb{R}^{k\times 2}; \quad l_{cc}^{r_1,r_2} = \mathbf{W_{cc}^\rho} l_{cc}^{r_1,r_2}, \mathbf{W_{cc}^\rho} \in \mathbb{R}^{k\times 2} \end{aligned} \tag{16}$$

where $\mathbf{W_h^\rho}$, $\mathbf{W_r^\rho}$, $\mathbf{W_b^\rho}$, $\mathbf{W_{as}^\rho}$, and $\mathbf{W_{cc}^\rho}$ are trainable parameters to transform the raw feature vectors into k-dimensional initial embedding. This uniform dimension of embeddings remains unchanged throughout the subsequent message-passing processes and enhances the representation learning on the graph. The edge embeddings are denoted without superscripts since they are not updated during message-passing after this linear projection. Based on the initial node embeddings and fixed edge embeddings, the aforementioned message-passing procedure will be executed for J iterations. In an iteration, the first step is using inbound train nodes to update route nodes. Considering that the operation status of inbound trains, such as the primary delay time, whether being postponed, the remaining time until receiving, have varying significance for route availability, we apply the Graph Attention Network (GAT) that can adaptively assign weighs among neighboring nodes to capture these differences. The multi-head mechanism is also applied to GAT to stabilize the training process and improve generalization ability. The specific computation of updating a route node r is outlined below:

$$\begin{aligned} \bar{e}_{r,h}^f &= LeakyReLU(\mathbf{ATT}_{h\to r,f}^{(j)^\mathrm{T}}[\mathbf{W}_{r_1}^{(j)} v_r^{(j)} || \mathbf{W}_{h_1}^{(j)}[v_h^{(j)} || l_{as}^{h,r}]]), \\ \bar{e}_{r,r}^f &= LeakyReLU(\mathbf{ATT}_{h\to r,f}^{(j)^\mathrm{T}}[\mathbf{W}_{r_1}^{(j)} v_r^{(j)} || \mathbf{W}_{r_1}^{(j)} v_r^{(j)}], \end{aligned} \tag{17}$$

$$e_{r,h}^f = \frac{\bar{e}_{r,h}^f}{\bar{e}_{r,r}^f + \sum_{h\in\mathcal{V}_H^i\cap\mathcal{N}_{in}(r)} \bar{e}_{r,h}^f}, \quad e_{r,r}^f = \frac{\bar{e}_{r,r}^f}{\bar{e}_{r,r}^f + \sum_{h\in\mathcal{V}_H^i\cap\mathcal{N}_{in}(r)} \bar{e}_{r,h}^f}, \quad \forall h \in \mathcal{V}_H^i \cap \mathcal{N}_{in}(r) \tag{18}$$

$$\dot{v}_r^{(j)} = \sigma\left(\frac{1}{F}\sum_{f=1}^{F}\left(e_{r,r}^f \mathbf{W}_{r_1}^{(j)} v_r^{(j)} + \sum_{h\in\mathcal{N}_{in}(r)} e_{r,h}^f \mathbf{W}_{h_1}^{(j)}[v_h^{(j)} || l_{as}^{h,r}]\right)\right) \tag{19}$$

Eq. (17) first computes the attention coefficient between an inbound train node h and a route node r. j denotes the index of iteration. f denotes the index of attention head. The embedding of inbound train node h is concatenated with the embedding of the assignment edge between h and r as $[v_h^{(j)} || l_{as}^{h,r}] \in \mathbb{R}^{2k}$. Then two trainable linear transformations $\mathbf{W}_{r_1}^{(j)} \in \mathbb{R}^{k\times k}$ and $\mathbf{W}_{h_1}^{(j)} \in \mathbb{R}^{k\times 2k}$ are applied to the embeddings of route node r and the concatenated inbound train node, respectively. The projected embeddings are concatenated and fed into a single-layer feedforward neural network containing am attention weight $\mathbf{ATT}_{h\to r,f}^{(j)} \in \mathbb{R}^{2k}$ and $LeakyReLU$ activation. The attention coefficient of route node r to itself is also computed using the same procedure, except no edge embedding is included. $\mathbf{ATT}_{h\to r,f}^{(j)}$ is the f-th attention head. Then, all coefficients for a route node r are normalized together using the softmax function to finalize the normalized attention coefficients, as shown in Eq. (18), where $\mathcal{N}_{in}(r)$ is the set of incoming neighboring nodes of the route node r. Finally, as shown in Eq. (19), using the attention coefficients computed by a single attention head, we fuse the embedding of the route node itself and extended inbound train node embedding. The multi-head mechanism averages the embeddings generated

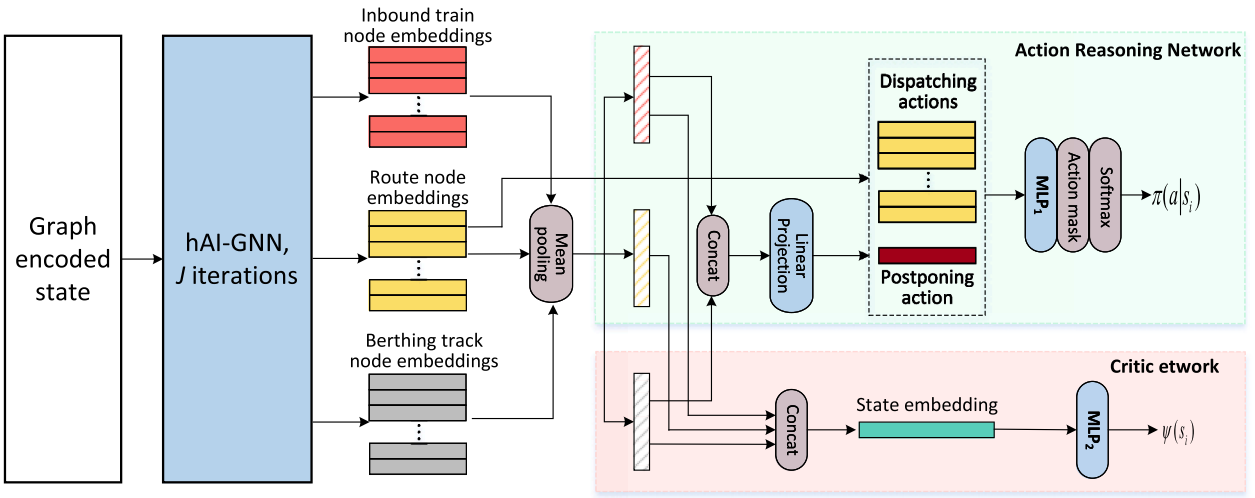


Fig. 7. The architecture of the policy network.

by all F attention heads. The embedding of route node r is finalized by the activation layer of a sigmoid function. With this attention based message-passing way, the importance of inbound trains can be considered. It should be noted that only receiving route nodes have been updated after the above node embedding computation process. This is because inbound train nodes are not connected to outbound route nodes, such a computation is not applicable if no assignment edge exists between two nodes. In addition, $\dot{v}_r^{(j)}$ obtained by Eq. (19) is an intermediate embedding of route node r in an iteration, since the route node embeddings will be updated in subsequent two message-passing steps, i.e., using berthing track nodes and route nodes themselves to update route nodes.

To make the route nodes be aware of information from berthing track nodes, we use Graph Isomorphism Network(GIN) to implement the message-passing, because only one berthing track node is connected to a route node. The specific node embedding update process is formulated as below:

$$\ddot{v}_r^{(j)} = \mathbf{MLP}_{b\to r}^{(j)}((1+\delta_{b\to r}^{(j)})\cdot \dot{v}_r^{(j)} + \sum_{b\in\mathcal{V}_B\cap\mathcal{N}_{out}(r)} v_b^{(j)}) \tag{20}$$

where $\mathcal{N}_{out}(r)$ includes outgoing neighboring nodes of route node r. In Eq. (20), we pass the sum of embeddings of neighboring berthing track nodes to the route node r, then integrate the sum into the intermediate embedding of route node r obtained by the previous step, where the trainable value $\delta_{b\to r}^{(j)}$ is used to automatically retain partial information from the route node r itself. Finally, a multi-layer perceptron $\mathbf{MLP}_{b\to r}^{(j)}$, which contains two hidden layers with $2k$ dimensions and a ReLU activation function, is applied to evolve the integrated embedding. The obtained route node embedding is still an intermediate one, which is used in the next message-passing step, loading route conflict information for route nodes. The GIN is also used in this step but modified as below to include conflict edge embeddings

$$v_r^{(j+1)} = \mathbf{MLP}_{r\to r}^{(j)}((1+\delta_{r\to r}^{(j)})\cdot \ddot{v}_r^{(j)} + \sum_{r'\in\mathcal{V}_R\cap\mathcal{N}(r)} (\ddot{v}_{r'}^{(j)} + l_{cc}^{r',r})) \tag{21}$$

where $\mathcal{N}(r)$ includes all neighboring nodes of route node r; $\mathbf{MLP}_{r\to r}^{(j)}$ and $\delta_{r\to r}^{(j)}$ are a multi-layer perceptron and a trainable value, respectively. We can see that the embeddings of neighboring nodes and corresponding conflict edges are summed, i.e., $(\ddot{v}_{r'}^{(j)} + l_{cc}^{r',r})$. After this step, for each route node, the information from inbound trains, outbound trains, and other routes are loaded to generate the node embedding. Then, the informative route node embeddings are used to update the embeddings for inbound train nodes and berthing track nodes in the last two steps. It is obvious that different routes have various importance to an inbound/outbound train or a berthing track. Hence, we turn back to using GAT to complete the message-passing and aggregation in the last two steps. The finalized node embeddings of inbound train nodes and berthing track nodes are denoted as $v_h^{(j+1)}$ and $v_b^{(j+1)}$, respectively. The equations are similar to Eq. (17) to (19) and are omitted here.

At this point, we have completed one iteration of node embedding, which is a layer in the hAI-GNN. All node embeddings on graph G_i are updated. We apply J iterations to deliver more advanced embeddings. That is, the hAI-GNN has J layers with the same neural structure but independent trainable parameters. Again, we keep all embeddings among different layers as k-dimensional. The finalized node embeddings generated by the hAI-GNN are $v_h^{(J)}, v_r^{(J)}, v_b^{(J)} \in \mathbb{R}^k$.

4.3.2. ARN For decision-making

Based on the node embeddings derived from the hAI-GNN, an action reasoning network (ARN) is proposed to make a decision, i.e., selecting a dispatching or postponing action for train h_i under state s_i. The overall policy network of our L2RP method, including hAI-GNN and ARN, is shown in Fig. 7.

The representations of actions are determined using node embeddings. Considering that the dispatching actions are selecting receiving and departure routes, they are directly represented by the k-dimensional route node embeddings, i.e., $v_r^{(J)}$ provided by hAI-GNN. On the contrary, the postponing action should be represented by information of trains, routes, and berthing tracks to consider the overview train operation status under a state, since it is used to reschedule trains with the purpose of minimizing the total knock-on delays. Therefore, we aggregate the node embedding of inbound trains and berthing track nodes to encode the postponing action as below:

$$v_{a^\omega} = \mathbf{W_x}[\frac{1}{|\mathcal{V}_H^i|}\sum_{r\in\mathcal{V}_H^i} v_h^{(J)} \| \frac{1}{|\mathcal{V}_B|}\sum_{b\in\mathcal{V}_B} v_b^{(J)}] \tag{22}$$

where v_{a^ω} is the embedding of the postponing action. Specifically, the embeddings of inbound train nodes and berthing track nodes are averaged respectively and then concatenated. To facilitate the uniform action evaluation, this $2k$-dimensional concatenated vector are linearly projected by $\mathbf{W_x} \in \mathbb{R}^{k\times 2k}$. As a result, the postponing action is also represented by a k-dimensional embedding. All embeddings of actions are fed into a multi-layer perceptron, as below,

$$f(v_a) = \mathbf{MLP}_1(v_a), \forall v_a \in \{v_r^{(J)} | r \in \mathcal{V}_R\} \cup \{v_{a^\omega}\} \tag{23}$$

where $\mathbf{MLP}_1$ is a multi-layer perceptron with two $2k$-dimensional hidden layers, a Tanh activation function, and a single-value output. The value is the score of selecting the corresponding action. Using the action mask stated in Eq. (B.1), the scores of infeasible actions are removed. Then the probabilities of selecting each remaining available action are computed by the Softmax function as

$$\mathcal{P}(a) = \frac{exp(f(v_a))}{\sum_{a'\in A_i} exp(f(v_{a'}))}, \quad a \in A_i \tag{24}$$

where $\mathcal{P}(a)$ denotes the probability distribution of all available actions under state s_i, which is used to sample an action. At this point, the policy network, including hAI-GNN and ARN, is established completely as $\pi_\theta(a|s_i) = \mathcal{P}(a)$, where π_θ is the policy network, θ represents all trainable parameters in the network.

4.4. Policy optimization

The DRL agent needs to collect experience (states, actions, and rewards) by repeated interaction with the environment. The purpose is to strengthen the policy network by learning effective actions under various states, and ultimately produce an outstanding policy for solving TPRP. This is the training process of the policy network.

In this paper, we train the policy network using Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017), which is an actor-critic algorithm. The actor refers to the policy network π_θ stated above. The critic is also a neural network in our L2RP method to estimate the cumulative rewards under each state, which is displayed in Fig. 7 and formulated as below,

$$\psi(s_i) = \mathbf{MLP}_2[\frac{1}{|\mathcal{V}_H^i|}\sum_{h\in\mathcal{V}_H^i} v_h^{(J)} \,||\, \frac{1}{|\mathcal{V}_R|}\sum_{r\in\mathcal{V}_R} v_r^{(J)} \,||\, \frac{1}{|\mathcal{V}_B|}\sum_{b\in\mathcal{V}_B} v_b^{(J)}] \tag{25}$$

where we first generate the $3k$-dimensional state embedding by concatenating the averaged node embeddings of inbound trains, routes, and berthing tracks. Then a multi-layer perceptron $\mathbf{MLP}_2$ is used to process the state embedding and estimate the cumulative reward. $\mathbf{MLP}_2$ has the same structure as $\mathbf{MLP}_1$ but a different input dimension ($3k$ instead of $2k$). Following the standard training procedure of PPO algorithm, we set L training iterations, and D instances of the TPRP are solved by the DRL agent in parallel in each training step. For each instance, the action a_i is sampled by π_θ and executed in the environment so that state s_i transits to the next state s_{i+1}. The training instances are replaced every E iteration. We use the general losses in PPO and update the neural networks by the Adam optimizer.

The action sampling measure is also applied in the validation and testing process after training. The solution delivered by sampling may be different each run due to the stochasticity, which is expected to find better solutions than only selecting the action with the highest probability. Because, in general, the current DRL algorithms can only converge to sub-optimal policies. To this end, in our L2RP method, D' copies of a single instance are solved in parallel when validating or testing the instance. The best solution among these copies is then selected as the final solution to the instance.

5. Experiments

In this section, the proposed L2RP method for the TPRP is tested on real-world instances. The experiment scenarios and settings are introduced in Section 5.1. The solution quality and generalization ability of our L2RP method are demonstrated in Sections 5.2 and 5.3, respectively. In Section 5.4, we implement an ablation study of different neural networks in the policy network. Finally, in Section 5.5, train delay scenarios of line disruptions are tested to further demonstrate the performance of our L2RP method when generalized to various train operation situations.

5.1. Experiment preparations

Experiment scenarios and schemes. The experiments are conducted using instances from two stations. The first is ZhengXi yard in Xi'an North high-speed railway station in China. It has 3 entry points, 3 exit points, 15 berthing tracks (with corresponding

platforms), and 77 routes. The other is JingGuang yard in Zhengzhou East high-speed railway station, which has 14 berthing tracks, 5 entry points, 5 exit points, and 94 routes. The schematic figures of these two yards are shown in Fig. D.1 and Fig. D.2, respectively. 5 real-world train timetables of Xi'an North station, named χ_{30}, χ_{100}, χ^1_{100}, χ^2_{100}, and χ_{200}, are used to be experiment scenarios. The specific settings of these timetables are listed in Table 3. Specifically, χ_{30} is a small-scale scenario covering 1.5 hours and 30 trains. When generating instances of this scenario, we randomly select 3–4 trains from the first 10 trains in the timetable to be the ones with primary delays, and the extent of each primary delay is sampled from a uniform distribution covering 8–20 minutes. χ_{100} is a large-scale scenario with 4.5 hours and 100 trains. Similarly, when generating instances, a random primary delay of 10–25 minutes is applied to each of 5–8 trains from the first 20 trains. χ^1_{100} and χ^2_{100} have the same operation horizon, the number of trains, and train delay settings as χ_{100}, but with different original schedules for the trains. Γ_{150} is another large-scale scenario of Zhengzhou East station. It covers 6.5 hours with 150 trains. When generating instances of this scenario, 5–10 trains from the first 25 trains are selected to have random primary delays of 10–25 minutes. These three large-scale scenarios are used for generalization experiments. χ_{200} is a huge-scale scenario covering 8.5 hours and 200 trains, used to set train delay scenarios caused by line disruptions. The specific train primary delays are determined according to the duration of line disruption and the minimum headway of 3 minutes.

Table 3
The experiment scenarios and primary delay settings.

Scenario	number of trains	planning period	number of trains with primary delay	primary delay ranges
χ_{30}	30	1.5 hours	3 ~ 4 trains	8 ~ 20 minutes
χ_{100}	100	4.5 hours	5 ~ 8 trains	10 ~ 25 minutes
χ^1_{100}	100	4.5 hours	5 ~ 8 trains	10 ~ 25 minutes
χ^2_{100}	100	4.5 hours	5 ~ 8 trains	10 ~ 25 minutes
Γ_{150}	150	6.5 hours	5 ~ 10 trains	10 ~ 30 minutes
χ_{200}	200	8.5 hours	determined by specific line disruptions	

We generate training, validation, and testing instances based on these timetables and delay settings. When training a DRL agent of our L2RP method under a scenario, training instances are generated on-the-fly during the training processes, and 50 validation instances are generated using the same timetable before training. Testing instances are generated according to the specific requirements of each experiment.

The following numerical experiments contain four parts of investigation respectively: ***(1) Testing performance:*** To demonstrate the solution quality, two DRL agents are trained based on small-scale and large-scale scenarios (χ_{30} and χ_{100}), respectively. ***(2) Generalization performance:*** To evaluate a trained DRL agent on timetables that it has never been trained with, we directly use the DRL agent trained on χ_{100} to solve testing instances of χ^1_{100}, χ^2_{100}, and Γ_{150} without retraining. ***(3) Ablation study:*** To validate the effectiveness of the penalty coefficient in the reward function, we conduct a sensitivity experiment with different values of the penalty coefficient. Also, to validate the effectiveness and advantages of the proposed hAI-GNN in the policy network, we conduct an ablation study with different neural networks replacing the hAI-GNN. ***(4) Line disruptions:*** To further test the generalization ability on different train delay scenarios, we set line disruptions to cause train primary delays under scenario χ_{200}, and still use the DRL agent trained on χ_{100} to solve instances.

Hyperparameters. The hyperparameters in our L2RP method are set as follows by configuration experiments. The number of message-passing iterations in the hAI-GNN is set as $J = 2$. The number of attention heads in the multi-head mechanism of hAI-GNN is set as $F = 8$. The uniform dimension in the policy network is set as $k = 64$. For the PPO algorithm, the total training step is set as $L = 500$. The number of paralleled training instances in each step is set as $D = 20$, while training instances are replaced every $E = 20$ training steps. The samples of 20 parallel instances are divided into mini-batches of size 64. The iteration of epochs for repeatedly updating parameters using these mini-batches is set to 3. The discount factor of the Generalized Advantage Estimation function is set to 0.99. The ratio of the gradient clip in the loss function is set to 0.2. The coefficients of the policy loss, value loss, and entropy term are set to 1, 0.5, and 0.01, respectively. The learning rate is set to 0.0002, with a decay frequency of every 50 training steps and a decay factor of 0.95. The discount factor for rewards is set to 0.97. The penalty for platform change in the reward function is set as $\sigma = 200$. For solving a testing instance, the number of copies is set as $D' = 50$.

Benchmarks and evaluation. In general, to quantitatively demonstrate the solution quality, we compute the performance gap for each individual instance between a given solution and the benchmark solution using the following formula:

$$Gap = \frac{(\Delta + Obj_x) - (\Delta + Obj_{bm})}{\Delta + Obj_{bm}} \times 100\,\% \tag{26}$$

Here, Δ denotes the total primary delay time of trains in the instance, used to avoid inconsistent gap values when dealing with negative or small objective function values; Obj_x and Obj_{bm} represent the objective value (defined in Eq. (4)) obtained by the method under evaluation and the benchmark method, respectively, which are subject to experiments and scenarios.

In the ***Testing Performance*** experiment, for the small-scale scenario (χ_{30}), we use CPLEX to solve the mixed integer linear program model (MILP) of the instances as the benchmark to evaluate our L2RP method. For the large-scale scenario (χ_{100}), a Rolling-Horizon Algorithm (RHA) is used as the benchmark, since CPLEX cannot obtain meaningful solutions in a reasonable solving time. We set the rolling scheme in the algorithm as [5, 15, 1800], which means that each rolling iteration solves a subproblem covering 15 trains within a time limit of 1800 seconds and finalizes the results for the first 5 trains. The MILP model and the rolling-horizon algorithm can be found in the previous work (Lu et al., 2022). Moreover, we also include the Greedy Interchange Heuristic Algorithm (GIHA)

Table 4
The experiment results of solving small-scale instances by L2RP and benchmarking against CPLEX.

Instance		CPLEX			L2RP		Gap between CPLEX and L2RP
Index	Total primary delay (sec)	Obj	Gap between upper and lower bound	Solving time (sec)	Obj	Solving time (sec)	
1	3000	−26	0 %	30.82	4	6.62	1.01 %
2	2100	−23	0 %	75.68	−16	5.36	0.34 %
3	2160	62	0 %	109.24	70	5.59	0.36 %
4	1920	717	0 %	202.40	756	5.61	1.48 %
5	2820	242	0 %	1316.45	289	5.54	1.53 %
6	1980	1364	0 %	446.18	1364	5.43	0 %
7	2760	115	0 %	1855.27	115	5.99	0 %
8	2400	976	0 %	701.91	1024	6.56	1.42 %
9	3480	1757	3.42 %	3600	1963	5.44	3.93 %
10	3180	2007	5.11 %	3600	2092	6.44	1.64 %
11	2160	2686	5.10 %	3600	2710	6.23	0.50 %
12	3240	1567	4.45 %	3600	1592	5.68	0.52 %
13	2460	2421	3.72 %	3600	2550	5.89	2.64 %
14	3720	2170	5.12 %	3600	2287	5.84	1.99 %
15	2940	2412	4.14 %	3600	2533	6.61	2.26 %

introduced by García-Ródenas et al. (2024) as an additional baseline. GIHA solutions are also compared against the RHA benchmarks to demonstrate the difference between our L2RP method and such a modern heuristic approach.

In the ***Generalization Performance*** experiment, for the three generalization scenarios (χ^1_{100}, χ^2_{100}, and Γ_{150}) we also demonstrate the solution quality of our L2RP method against the same RHA benchmark.

Specially, in the ***Line disruptions*** experiment, the benchmarking is conducted based on the First-Come-First-Service (FCFS) heuristic rule, and we directly compare the train operation recovery times of L2RP solutions and FCFS solutions, which are detailed in Section 5.5.

All experiments, including the training and testing of L2RP, and all benchmark evaluations, are implemented on the same device, a PC with an Intel i7-14700KF CPU and a single Nvidia GeForce 4090 GPU. The solving times of all involved methods are also reported to compare their solving efficiency.

5.2. Testing performance

In this section, we demonstrate the performance of the proposed L2RP method for the TPRP, in terms of its solution quality and solving efficiency. We train two agents for the small and large-scale scenarios, respectively, based on the synthetic training instances, with validations every 50 iterations.

5.2.1. Small-scale instance

We first train a DRL agent on the small-scale scenario χ_{30}. After the training is completed, we choose the trainable parameters that obtain the best objective value on the validation instances to form the trained agent. Then, the trained DRL agent is used to solve 15 testing instances. The CPLEX is also used to solve these testing instances with a solving time limit of 60 minutes for each instance.

The objective values and solving time are shown in Table 4, together with the results provided by CPLEX. It can be seen that using CPLEX to solve the MILP model from instances 1 to 8 yields optimal solutions within the solving time limit of 60 minutes. But the solving time has a huge variance due to various amounts of route conflicts to solve. From instances 9 to 15, the CPLEX outputs a feasible solution at the upper bound, since it cannot reach the optimality with the solving time limit. The gaps between the upper and lower bound illustrate that the feasible solutions are close to optimal.

Using our L2RP method, the solving time of each instance is around 6 seconds, which is much faster than CPLEX and remains stable despite the varying amounts of route conflicts encountered across different instances. Meanwhile, the gaps between the L2RP solutions and optimal solutions provided by CPLEX show that the L2RP method can obtain near-optimal solutions. For example, from instance 1 to instance 8, the gaps between L2RP solutions and CPLEX solutions are lower than 1.53 %, which means the L2RP solutions are very close to optimal solutions. Especially for instances 6 and 7, the L2RP method can also obtain the optimal solution, as the gaps are 0 % shown in the table. On the other hand, for instance 9 to instance 15, the gaps are calculated by the L2RP solutions and the upper bound solutions of CPLEX. The maximum gap is only 3.93 %, and others are lower than 2.64 %. Such small gaps indicate that the L2RP solutions are very close to the feasible solutions at the upper bounds of CPLEX. Especially in instances 11 and 12, the L2RP method can obtain solutions with gaps lower than 1 %. Considering such feasible solutions are near-optimal (as the gaps between upper and lower bounds are only around 3∼5 %), we can say that our L2RP solutions are still of high quality. In conclusion, using the proposed L2RP method to solve small-scale instances of TPRP, we can obtain superior solutions in a very short solving time.

5.2.2. Large-scale instance

To test the L2RP method in solving large-scale instances, we train a new DRL agent under the scenario χ_{100}, i.e., 100 trains in 4.5 hours. During the training process, the agent with the best performance on the validation instances is saved and then used to solve

Table 5
The experiment results of 50 large-scale instances.

	Average solving time (sec)	Average Obj	Average Gap
L2RP	21	1501	3.38 %
GIHA	13	1803	7.92 %
RHA	1889	1229	—

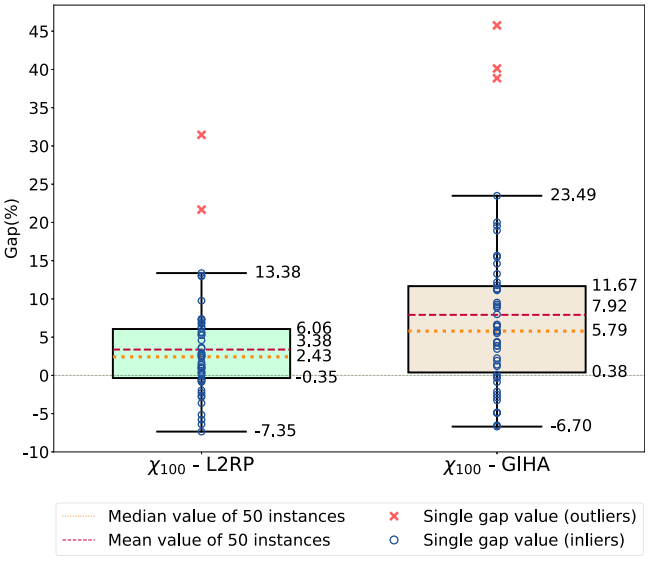


Fig. 8. The gap distribution of 50 large-scale instances.

50 testing instances. As previously stated, the performance of our L2RP method is evaluated by comparing it with the GIHA method, using the RHA method as the benchmark. The overall results across 50 testing instances are displayed in Table 5, including the average solving time and objective value of these three methods. The gaps between the RHA benchmark and the other two methods are computed according to the definition in Eq. (26).

It is evident that the L2RP method is significantly faster than the RHA method in solving large-scale instances (average 21s $\ll$ 1889s). Moreover, the range of solving time of our L2RP method is [19.89, 22.54] seconds, which means the solving time is still very stable. The key to achieving short and stable solving times under large-scale instances lies in the design of the graph-encoded state. The graph incorporates a fixed number of inbound train nodes, route nodes, and berthing track nodes. As a result, large-scale instances covering more trains do not lead to significantly more complex graph structures. Therefore, the DRL agent trained under the large-scale scenario has the same size of the policy network as the one trained under the small-scale scenario. Although more trains bring more decision steps required to be completed by the DRL agent in a large-scale instance, the computation cost for each step is almost the same as solving a small-scale instance. Hence, it is foreseeable that the solving time will increase linearly with the number of planned trains in the timetable rather than increasing sharply.

The solutions obtained by the L2RP method also demonstrate high quality. The average gap between L2RP and RHA solutions is only 3.38 %. We illustrate the distribution of gaps for 50 testing instances in a box plot, as shown in Fig. 8. There are 14 instances where the solution gap is negative, indicating the L2RP solution outperforms the RHA benchmark. Meanwhile, in three-quarters of testing instances, the L2RP method can obtain solutions with gaps less than 6.06 % compared to the RHA solutions. We note that the 2 outliers (with gaps larger than 20 %) indicate the trained DRL agent may not be adaptable enough for some special train delay situations.

The solution quality of the L2RP method surpasses the GIHA method. As shown in Table 5, the average objective value of GIHA solutions is 1803, which is noticeably higher (i.e., worse) than that of L2RP solutions. Similarly, the average gap of GIHA solutions relative to RHA solutions is 7.92 %, which is also higher than the average gap of L2RP (3.38 %). The gap distribution of GIHA solutions, which is also illustrated in Fig. 8, further reinforces this advantage. Although the GIHA method achieves 10 instances with negative gaps, slightly less than L2RP in this aspect (13 instances), its third quartile gap is 11.67 %, which is significantly larger than that of L2RP (6.06 %). Moreover, the three outliers larger than 35 % and a wider range of gaps indicate that the GIHA method exhibits higher performance fluctuation compared to our L2RP method. Based on these observations, it can be concluded that, while the GIHA method is also a sub-minute solver, even faster than L2RP, our L2RP method has overall better performance in solving TPRP.

Table 6
The experiment results for three generalization scenarios, with 50 instances in each scenario.

Scenario	Method	Average solving time (sec)	Average Obj	Average Gap
χ^1_{100}	L2RP	21	1136	4.52 %
	RHA	1746	820	—
χ^2_{100}	L2RP	21	4036	5.10 %
	RHA	2339	3535	—
Γ_{150}	L2RP	33	3223	5.06 %
	RHA	2930	2767	—

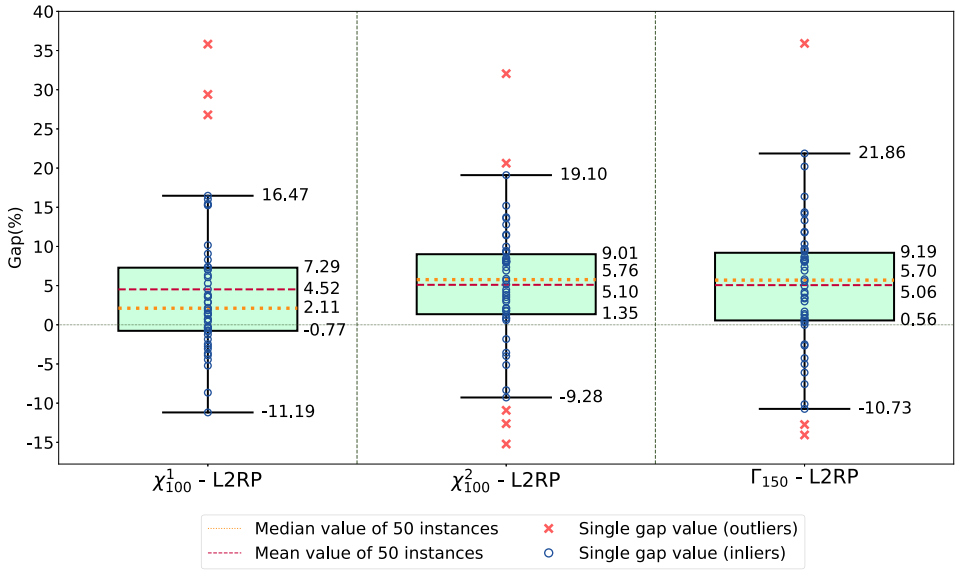


Fig. 9. The gap distribution of 50 instances for each generalization scenario.

In summary, the proposed L2RP method is capable to produce high-quality solutions for large-scale instances within a very short solving time. The demonstrated performances enable the L2RP method to be applied in real-world operations to provide real-time scheduling suggestions for dispatchers to cope with train delays.

5.3. Generalization performance

In the previous large-scale experiments, the training and testing instances are all generated under the same timetable scenario, which means that the DRL agent solves instances with the familiar timetable pattern (i.e., the same original train arrival and departure time) during training and testing. To access the generalization capability of the proposed L2RP method to unseen timetable scenarios and even different stations, in this section, the trained agent for the large-scale scenario χ_{100} is applied directly to solve the instances from another three large-scale scenarios without retraining. Specifically, two new large-scale scenarios χ^1_{100} and χ^2_{100} are also timetables covering 100 trains over 4.5 hours, respectively. But the scheduled arrival time, berthing track, and departure time of trains are completely different from χ_{100}. Meanwhile, most trains in χ^1_{100} have only a few minutes of scheduled dwell time, while χ^2_{100} has more trains with longer scheduled dwell time, leading to less slack time in the timetable. The third scenario Γ_{150} is a larger timetable covering 150 trains in 6.5 hours of another station, as we introduced in Section 5.1. For each of these three scenarios, 50 testing instances are generated. The same DRL agent trained in Section 5.2.2 is directly used to solve all testing instances without retraining. Generalization performance is evaluated by comparing our L2RP method with the RHA benchmarks. The experiment results, including average objective values, average gaps, and solving times, are summarized in Table 6 and Fig. 9.

We illustrate the solution quality of our L2RP method relative to the RHA benchmarks. Compared to the average gaps of 3.38 % achieved on scenario χ_{100}, the average gaps under the new scenarios increase slightly (4.52 %, 5.10 %, and 5.06 %). Nevertheless, these average gaps remain within an acceptable range, indicating that the agent trained on χ_{100} generalizes favorably to these unseen scenarios. More specifically, for scenario χ^1_{100}, the minimum and medium gaps of 50 testing instances are -11.19 % and 2.11 %, respectively, which are both smaller than those of χ_{100} (-7.35 % and 2.43 %). Moreover, the L2RP method outperforms the RHA benchmarks in 17 instances, better than its performance on scenario χ_{100}. Similarly, in scenario χ^2_{100} and Γ_{150}, the gap distributions demonstrate that our L2RP method can obtain solutions with a gap below about 9 % in three-quarters of instances. Although this represents an increase from 6.06 % observed in scenario χ_{100}, the minimum gaps and the number of instances with negative gaps are

Table 7
Sensitivity analysis results for the penalty coefficient σ on platform changes.

σ	Average obj	Average number of platform changes
0	1885	19.58
1	1888	19.51
25	1809	18.00
50	1703	17.42
100	1539	15.56
200	1501	15.58
300	1533	15.58
400	1520	15.60
500	1541	15.48

competitive with or even better than the scenario χ_{100}, confirming the agent trained on χ_{100} retains the ability to produce high-quality solutions under these generalization scenarios.

As shown in Table 6, the average solving time of our L2RP method remains substantially shorter than that of RHA benchmarks across all generalization scenarios. For timetables χ^1_{100} and χ^2_{100}, which have the same number of 100 trains as χ_{100}, the total number of decision steps to be completed by the DRL agent is similar to that in solving instances of χ_{100}. The graph encoded state ensures the computation cost at each step is the same, as we stated in Section 5.2.2. Hence, the average solving time remains consistent at 21 seconds. For the larger scenario Γ_{150}, the average solving time Meanwhile, due to more trains in scenario Γ_{150} and more routes in that station, the average solving time of Γ_{150} increases to 33 seconds due to the greater number of trains and routes. The solving time ranges for χ^1_{100}, χ^2_{100}, and Γ_{150} are [19.54, 22.89], [19.66, 22.61], and [31.56, 35.89] seconds, respectively, demonstrating stable and predictable solving time. In contrast, the RHA method shows highly variable and often prolonged solving times. For example, despite having the same number of trains, the RHA method requires more time to solve instances of timetable χ^2_{100} than timetable χ^1_{100} (2339 seconds > 1746 seconds). The reason is that more route conflict constraints have to be resolved in solving instances of a timetable with less slack time. The solving time ranges using the RHA method for the three scenarios χ^1_{100}, χ^2_{100}, and Γ_{150} are [463.54, 5623.46], [681.69, 6031.58], and [1109.78, 6037.01] seconds, respectively, demonstrating its volatile and instance-dependent efficiency. Therefore, it is worth emphasizing that our L2RP method can ensure very stable and short solving times if the number of trains in a timetable is given, regardless of train scheduled operation times and conflict situations.

Here we need to clarify that although high-quality solutions can still be delivered in these generalization experiments, we recommend retraining a new DRL agent for a new timetable. Because in real-world daily operations, the train timetable will not change frequently. Before a new timetable is put into practical operations, training a new DRL agent is an offline task requiring a couple of hours, which is acceptable at the planning stage of train timetabling. The specialized new agent may better capture the characteristics of scheduled train operations in the new timetable and thus output good solutions in real-time applications.

5.4. Ablation study

5.4.1. Penalty coefficient of penalty change

In the reward function, we set a penalty coefficient σ for the platform changes, as formulated in Eq. (15)a. It guides the DRL agent to adhere to the original platform schedule as much as possible, thereby aligning with the secondary optimization purpose defined in the objective function (Eq. (4)). Hence, in this section, to validate this design's effectiveness, we conduct a sensitivity analysis on the penalty coefficient σ. Specifically, a series of values {0, 1, 25, 50, 100, 200, 300, 400} is set for σ, where $\sigma = 0$ means platform changes are not penalized. Under each value, a new DRL agent is trained and then tested on the same set of 50 testing instances of χ_{100} used in Section 5.2.2. The average objective value and the average number of platform changes are listed in Table 7.

The experiment results in Table 7. demonstrate that the penalty coefficient σ is crucial in determining the solution quality.

On the one hand, the learned policies exhibit strong robustness when $\sigma \geq 100$. The average objective value stabilizes between 1500 and 1540, while the average number of platform changes remains around 15.5~15.6. This indicates that a sufficiently large penalty for platform changes can effectively guide the DRL agent to learn a stable policy for minimizing train knock-on delays while largely preserving the original platform assignments.

On the other hand, although the policy network can still converge with no penalty or a small penalty coefficient ($\sigma \leq 50$), the resulting performance is markedly inferior. For example, when the penalty coefficient is set to $\sigma = 0$ and $\sigma = 1$, the average objectives are 1885 and 1888, respectively, and the average number of platform changes is 19.58 and 19.51, respectively. This performance degradation occurs because a small σ value could be negligible compared to the delay penalties, which could reach hundreds or thousands of seconds. Hence, the DRL agent receives insufficient incentive to prioritize the original scheduled platforms, leading to excessive exploration of platform reassignments. Although the policy networks can eventually learn to avoid most detrimental platform changes, it is actually a result of incurring additional knock-on delays by platform changes, not the penalty coefficient itself. Since no positive reward is given for platform changes, any change only leads to more potential conflicts and additional delays, which encourages the DRL agent to prefer scheduled platforms to avoid the delay penalty. As a consequence, the small penalty of platform changes merely leads to less efficient learning processes and inferior policies.

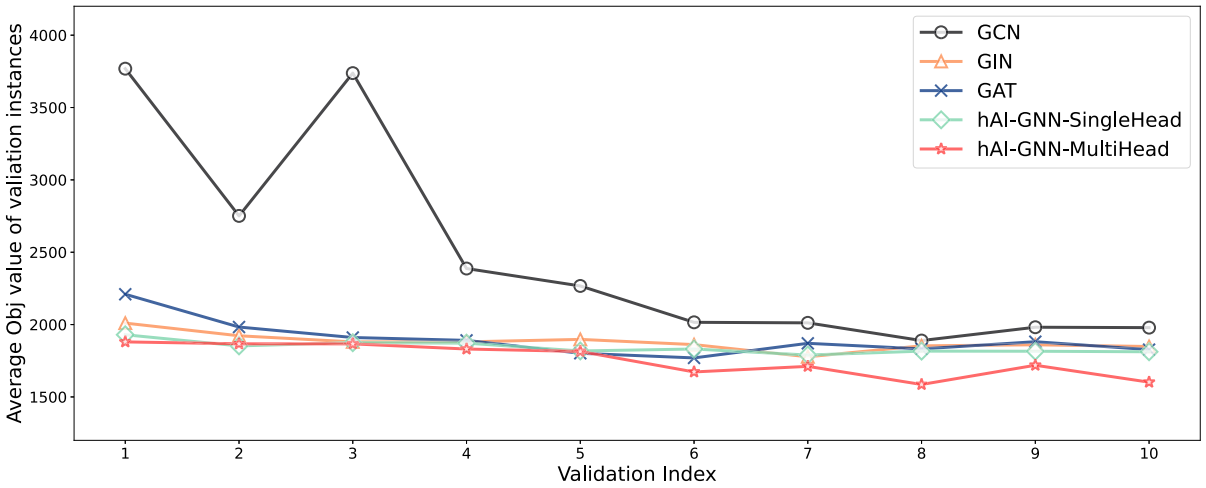


Fig. 10. The validation curves on scenario χ_{100} using different neural components.

Table 8
The ablation results of average gaps.

	Testing scenario		Generalization scenario		
	χ_{30}	χ_{100}	χ^1_{100}	χ^2_{100}	Γ_{150}
GCN	3.24 %	7.71 %	8.04 %	13.53 %	10.28 %
GIN	3.16 %	7.47 %	8.01 %	12.64 %	9.55 %
GAT	2.77 %	4.91 %	6.67 %	12.11 %	8.61 %
hAI-GNN-SingleHead	**1.25 %**	5.3 %	6.28 %	9.57 %	7.00 %
hAI-GNN-MultiHead (Our method)	1.29 %	**3.38 %**	**4.52 %**	**5.10 %**	**5.06 %**

Based on these findings, we select $\sigma = 200$ as the finalized penalty coefficient, as it yields the lowest average objective function value on the testing instances.

5.4.2. Neural network architecture

In the proposed policy network, the generated node embeddings from graph encoded state are essential for action reasoning in each decision step. Hence, the effectiveness of node embedding learning is significant for our L2RP method. In this section, we conduct an ablation study using various popular graph neural networks to prove the effectiveness and advantages of the proposed hAI-GNN.

Specifically, we set 4 different graph neural networks to replace the proposed hAI-GNN in the policy network to complete the designed message-passing procedures and generate node embeddings. Among them, 3 common and powerful graph neural networks, Graph Convolutional Network (**GCN**), Graph Isomorphism Network (**GIN**), and Graph Attention Network (**GAT**), are tested. Additionally, we cancel the multi-head attention mechanism in the proposed hAI-GNN, denoted as **hAI-GNN-SingleHead**, to be the last graph neural network for comparison. The hAI-GNN of our L2RP method stated above is denoted as **hAI-GNN-MultiHead**, which has 8 attention heads.

Using the 5 policy networks constructed with these graph neural networks for node embedding, we repeat the previous numerical experiments in Sections 5.2.1, 5.2.2 and 5.3, respectively, i.e., (1) training and testing on small-scale scenario χ_{30}, (2) training and testing on large-scale scenario χ_{100}, and (3) generalization testing on the other three large-scale scenarios χ^1_{100}, χ^2_{100} and Γ_{150}.

We first show the training processes under the large-scale scenario χ_{100}. For each policy network, the validation curve is plotted based on the average objective value of validation instances during the training process, as shown in Fig. 10. The validation frequency is every 50 steps, so 10 validation values are included in each curve. We observe that all 5 policy networks with different neural components converge after 300 training steps. However, the policy network using the hAI-GNN with the multi-head attention mechanism, i.e., the finalized structure in our L2RP method, outperforms other policy networks after convergence. It obtains the minimal average objective value of validation instances after 400 training steps.

We also illustrate all experiment results using 5 different policy networks in Table 8. These experiment results are average gaps between the solutions of testing instances provided by ablation methods and benchmark methods. We recall that for scenario χ_{30}, 15 testing instances are set, and CPLEX solutions are benchmarks; for scenarios χ_{100}, χ^1_{100}, χ^2_{100}, and Γ_{150}, 50 testing instances are generated, respectively, and RHA solutions are benchmarks for computing average gaps. It can be seen that policy networks constructed by the proposed hAI-GNN have better performance than policy networks with GCN, GIN, and GAT on testing instances of all 5 scenarios, regardless of whether the multi-head attention mechanism is applied. This is because hAI-GNN is specially designed for TPRP, which can capture effective features according to trains, routes, and berthing tracks. Moreover, the comparison between hAI-GNN-

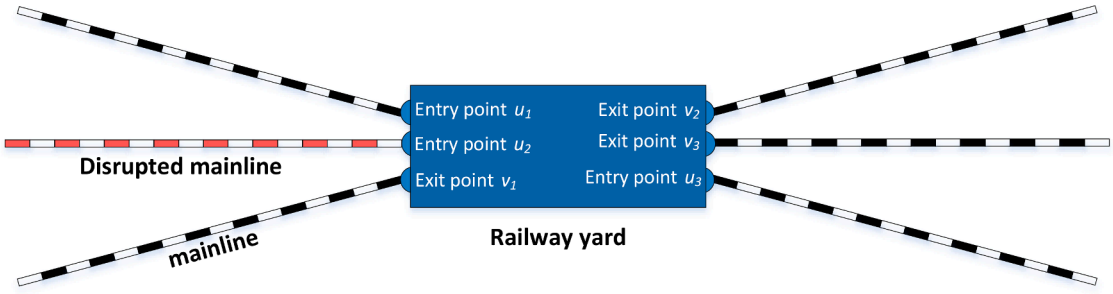


Fig. 11. The schematic diagram of the line disruption.

SingleHead and hAI-GNN-MultiHead shows that, the policy network using hAI-GNN without the multi-head attention mechanism can obtain high-quality solutions for the small-scale scenario, even a little better than hAI-GNN-MultiHead, i.e., 1.25 % < 1.29 %. However, the multiple attention heads in hAI-GNN plays a significant role in improving performance for large-scale scenarios, as shown in Table 8, where minimal average gaps for scenarios χ_{100}, χ^1_{100}, χ^2_{100}, and Γ_{150} are obtained by hAI-GNN-MultiHead. In summary, the proposed hAI-GNN with the multi-head attention mechanism outperforms other neural components, delivering effective node embeddings to obtain high-quality solutions.

5.5. Generalization for line disruptions

In the above experiments, the train primary delays are set randomly using given uniform distributions. Hence, this section aims to further validate the applicability of a trained DRL agent in more natural train delay scenarios. Specifically, consider a disruption in the line segment between the railway station and its predecessor, and the railway line links an entry point of the railway station. Then, all trains arriving from this line can not enter the line segment to approach the railway station. After the disruption is fixed, dispatchers will organize the affected trains on this line to operate according to the minimum headway to recover train operations to the original timetable as soon as possible (assuming no delayed train is canceled). Therefore, the primary delays of trains arriving from the line are determined according to the duration of disruption and the headway. In this case, the delay characteristic with dense arriving trains after the disruption is totally different from random delays among trains in the previous experiments.

The specific experiment settings in this section are as follows. As mentioned before, timetable χ_{200} with 200 trains in 8.5 hours is the experimental scenario. The disruption is set on the line that links to the second entry point of the railway station, as shown in Fig. 11. A line disruption begins at 02:00, supposing the planning period of timetable χ_{200} is 00:00–08:30. We conduct experiments of 3 instances with different disruption durations, i.e., 45 minutes, 60 minutes, and 75 minutes. For implementation convenience, the train running time in the segment between the railway station and its predecessor is included in these durations. That is, the scheduled arrival time of the first train delayed by the line disruption is 02:45 after the 45-minute disruption duration. The segment headway is set to 3 minutes. The scheduled arrival times of all delayed trains are accordingly determined.

We use a First-Come-First-Service (FCFS) heuristic rule for benchmarking in this section since previous experiments have demonstrated the RHA method needs a considerably long solving time to solve large-scale instances, which is not able to rapidly deal with numerous route conflicts in these real-world application scenarios. Specifically, at the scheduled arrival time of an inbound train, if its scheduled platform is not occupied, it will be received to arrive on its scheduled platform; otherwise, the FCFS rule will assign another available platform for it randomly, ensuring it can arrive at its scheduled arrival time. If no available platforms can be assigned, it will be held until any platform can be assigned for it. Similarly, after the dwell time, the FCFS rule will let the outbound train depart immediately if its departure route is available. The FCFS rule is implemented also using the simulation model mentioned in Section 4.1.3. Considering the total primary delay of trains could be up to tens of thousands of seconds in these line disruption scenarios, it could lead to unfair small gaps calculated by Eq. (26). Hence, for benchmarking between the L2RP solutions and FCFS solutions, we turn to define another evaluation indicator, named *train operation recovery time*. It can be calculated as

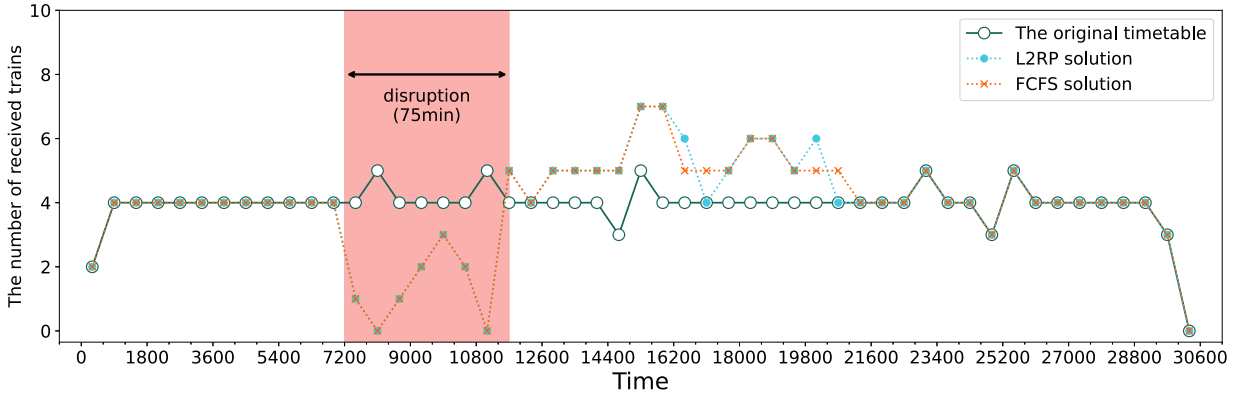
$$Recovery\ time = \tau^d_{h'} - I_{end}, \tag{27}$$

where h' is the last train departing with a departure time deviation, $\tau^d_{h'}$ is train h''s rescheduled departure time, I_{end} is the end time of the disruption. That is, the train operation recovery time is determined by the moment after which all trains can run according to the original timetable.

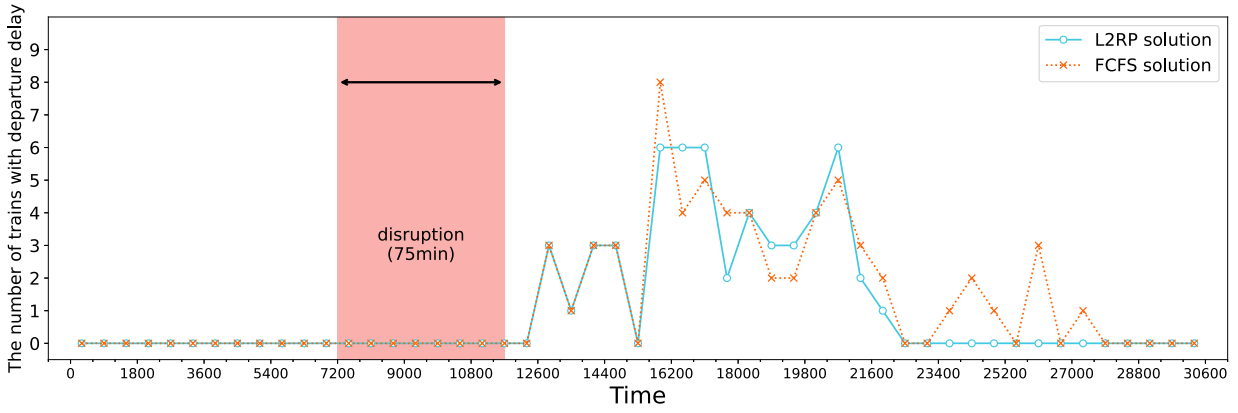
In order to verify the generalization ability of our L2RP method, we still use the DRL agent trained for the scenario χ_{100} in Section 5.2.2 to directly solve the 3 line disruption instances without retraining. The overall experiment results are shown in Table 9. Besides the train operation recovery time, we also include the objective value defined in Eq. (4) to show the difference of total knock-on delays between the L2RP solutions and FCFS solutions. According to the recovering time and objective value, it is evident that the trained DRL agent outperforms the FCFS rule in all 3 instances with different train delay conditions. Especially, in the two

Table 9
The experiment results of the line disruption instances, using L2RP method and benchmarking against FCFS.

Disruption scenario	The number of trains with primary delay	L2RP			FCFS		
		Obj	Recovery time (min)	Solving time (sec)	Obj	Recovery time (min)	Solving time (sec)
45 min (02:00–02:45)	35	3411	120.5	43.22	6140	147.9	0.52
60 min (02:00–03:00)	43	4612	141.7	44.89	5823	218.1	0.53
75 min (02:00–03:15)	49	6305	165.3	44.51	9851	257.5	0.51



(a) The number of received inbound trains, counted every 10 minutes



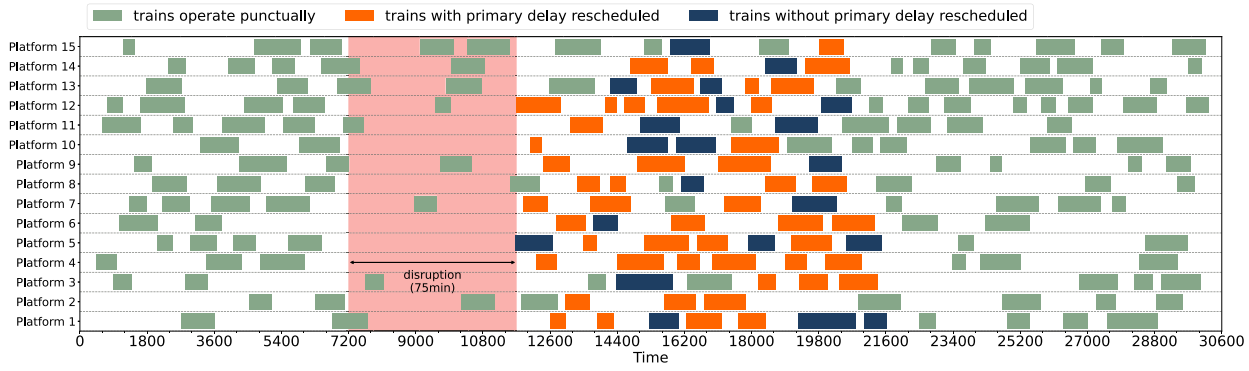
(b) The number of departed outbound trains with knock-on delays, counted every 10 minutes

Fig. 12. The traffic and delay statistic of the L2RP solution and FCFS solution of the 75-minute disruption instance.

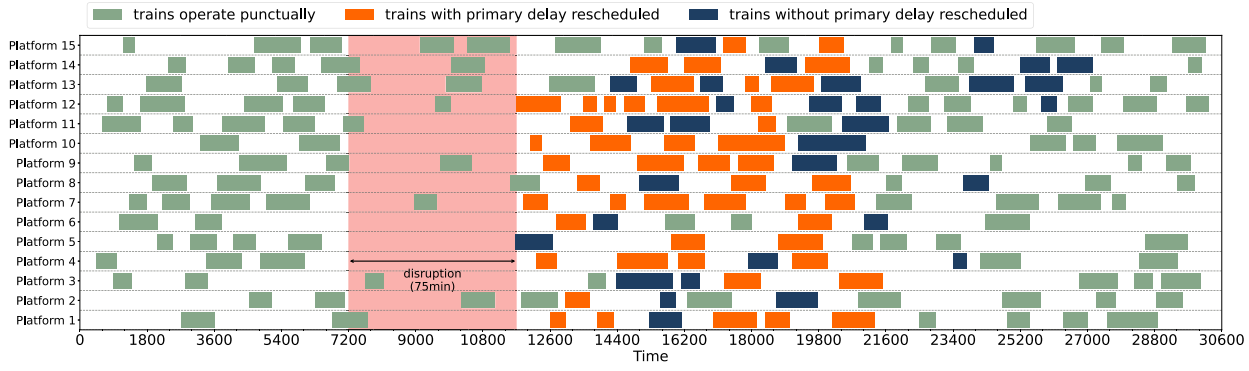
instances with 60-minute and 75-minute disruptions, the DRL agent takes 141.7 minutes and 165.3 minutes, respectively, to recover train operations to the original timetable, which are about 76 minutes and 92 minutes less than FCFS solutions. Such significant saved recovery time is very valuable for both passengers and dispatchers in real-world train operations. On the other hand, the solving time using the DRL agent is about 44 seconds, which is about double compared to instances with 100 trains (i.e., 21 seconds in Tables 5 and 6). The advantage of solving time only increasing with the number of trains is again demonstrated. Although the solving time of the FCFS rule is extremely short, the solution quality is not acceptable due to the quite long recovery time and extensive knock-on delays.

Using the 75-minute disruption instance as an example, we further illustrate how the L2RP solution and FCFS solution affect the train operation details in the railway station.

Firstly, the train traffic in the railway station is counted every 10 minutes according to the train actual arrival time, as shown in Fig. 13a. Compared to the original timetable, the overall train traffic in the railway station presents a general trend of first decreasing due to disruption, then increasing to higher than normal due to rescheduling, and finally returning to normal. Such a



(a) The L2RP solution



(b) The FCFS solution

Fig. 13. The gantt chart of platform utilization of the 75-minute disruption instance.

trend exists both in the L2RP solution and FCFS solution. The FCFS rule aims to receive trains using available platforms as much as possible, the traffic volume is therefore increased to a higher level than the original timetable after the disruption. Even so, the DRL agent aiming to minimize knock-on delays instead of only focusing on receiving trains still returns the traffic volume to normal one unit (i.e., 10 minutes) earlier than FCFS. Therefore, although the DRL agent is trained under another scenario based on the main purpose of minimizing the total knock-on delays, it can leverage the railway station capacity to recover train operations after the disruption, and shows better utilization than the FCFS rule. It is noted that during the disruption, the numbers of received inbound trains in the two solutions are identical, since trains arriving from other mainlines without disruption can still operate punctually.

To investigate the distribution of trains with knock-on delays in the planning period, we count the outbound trains with departure knock-on delays according to train actual departure time, considering the arrival knock-on delay is included in the departure knock-on delay of each train. The statistic unit is also 10 minutes. The results of the L2RP solution and FCFS solution are shown in Fig. 13b. It can be observed that all trains operate punctually before the disruption. After the disruption ends, the number of trains with knock-on delays begins to increase, since the trains delayed by the disruption begin to arrive at the station. The numbers of trains with knock-on delays in the L2RP solution and FCFS solution in each 10-minute time unit have no significant difference, because the dense arrival of trains after the disruption leads to numerous route conflicts and knock-on delays. However, it is obvious that the L2RP solution returns the number of departed outbound trains with knock-on delays to 0 about 90 minutes earlier than the FCFS solution, as we reported in Table 9.

We finally illustrate the platform utilization of the 75-minute disruption scenario using Gantt charts in Fig. 13, including the L2RP solution and FCFS solution. In each subplot, we highlight the rescheduled trains and distinguish them into two categories according to whether a delayed train has a primary delay. The orange bars indicate trains that have primary delays, i.e., their scheduled arrival /departure times have been changed due to the line disruption. The deep blue bars indicate trains that only have knock-on delays; that is, the line disruption has not changed their scheduled arrival/departure time, but these trains are still rescheduled to match the overall train platforming and rescheduling. In Fig. 13a of the L2RP solution, the departure time of the last train with primary delay and the last train with only knock-on delay are almost the same, i.e., around 06:00 at Platform 3 and Platform 1, respectively. On the contrary, in Fig. 13b of the FCFS solution, after all trains with primary delays are resolved, multiple trains are still rescheduled to operate with knock-on delays in the succeeding period of about 90 minutes. Meanwhile, the total number of trains that only have knock-on delays in the L2RP solution is 21, which is increased to 31 in the FCFS solution. In conclusion, the trained DRL agent is able

to recover train operations as soon as possible and simultaneously reduce the resulting knock-on delays to other trains not affected by the line disruption.

6. Conclusion

In this paper, we propose a novel deep reinforcement learning method, named Learning to Reschedule Platforming (L2RP), to address the Train Platforming and Rescheduling Problem (TPRP) while ensuring solution quality, efficiency, and precision. The core of our method lies in integrating microscopic simulation with graph neural network based deep reinforcement learning to achieve intelligent decision-making for TPRP. In this method, we formulate a Markov decision process (MDP), in which a graph encoded state captures intricate interdependencies among trains, routes, and berthing tracks. A microscopic simulation model of train operations in the station is developed to not only integrate the theoretical MDP model and the decision-making agent (i.e., the specialized graph neural network based policy network) by providing states, executing actions, and completing transitions, but also ensure fine spatial and temporal precision in solutions.

The experimental results demonstrate that, for the TPRP with different scales, the proposed L2RP method consistently delivers high-quality solutions comparable to or superior to those obtained by conventional optimization and heuristic approaches. Meanwhile, it exhibits strong generalization capabilities in handling unseen timetables, various delay scenarios, and severe line disruptions. Moreover, the computational time of our method remains very low and stable under the TPRP instances with the same scale, regardless of the delay scenario, and only increases linearly in solving larger-scale instances, making it highly promising for real-time applications.

To summarize, the proposed L2RP method contributes in several ways:

- We propose a novel method for solving TPRP by combining the policy learning capabilities of deep reinforcement learning, the representation learning power of graph neural networks, and the high modeling precision of microscopic simulation. This integrated framework provides a promising solution paradigm for solving train platforming related problems or other train scheduling problems at the station level.
- Our L2RP method learns effective and powerful policies for solving TPRP, producing high-quality solutions for instances with different scales and various train delay scenarios. The learned policies also show robust performance when generalized to other timetables or even major disruption scenarios. Such outstanding solving ability and reliable generalization performance show good potential for application in real-world scenarios.
- Our L2RP method overcomes the traditional trade-off between the modeling granularity and the solution efficiency in solving TPRP, since the precision of train operations in solutions is guaranteed by employing the microscopic simulation model, and the solving time is consistently short and stable under various instances.

We close by pointing out several future research directions. The most obvious next step is that we expect to apply the deep reinforcement learning methods to solve the TPRP for multiple stations along a railway line or in a railway network, instead of studying only a single station. In addition, the penalty for platform changes in the objective and reward function can be customized for specific purposes, such as minimizing transfer distance in platform reassignments to enhance passenger experience. Furthermore, incorporating other constraints related to crew scheduling and rolling stock service connections would further improve the practical applicability and acceptability of the TPRP solutions. Lastly, the train routing problem, where more than one route can be used between a berthing track and a boundary point, may also be considered to be solved by deep reinforcement learning in the future.(Fig. 12).

CRediT authorship contribution statement

Hongxiang Zhang: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Data curation, Conceptualization; **Andrea D'Ariano:** Validation, Supervision, Formal analysis, Conceptualization; **Yongqiu Zhu:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization; **Yaoxin Wu:** Writing – review & editing, Methodology, Formal analysis; **Liuyang Hu:** Visualization, Validation, Software, Investigation, Data curation; **Gongyuan Lu:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Funding acquisition.

Data availability

The authors do not have permission to share data.

Acknowledgments

This work is supported by National Key Research and Development Program of China (No. 2022YFB4300504) and National Natural Science Foundation of China (No. 52272324). The first author is deeply grateful for the support from the China Scholarship Council, China (No. 202207000095).

Appendix A. Notations

Table A.1
The notations in the problem description and MDP model.

Notation	Description	Notation	Description
$\mathcal{H}$	Set of trains in the considered period, indexed by h.	r^p_h	Actual used receiving route of train h.
$\mathcal{B}$	Set of berthing tracks in the railway station, indexed by b.	r^q_h	Actual used departure route of train h.
$\mathcal{R}$	Set of routes in the railway station, indexed by r.	y_r	The travel time for a train traversing route r.
$\mathcal{R}^p$	Set of receiving routes in the railway station, $\mathcal{R}^p \subset \mathcal{R}$.	$\gamma_{r,r'}$	Degree of Conflict of route r to route r'.
$\mathcal{R}^q$	Set of departure routes in the railway station, $\mathcal{R}^q \subset \mathcal{R}$.	x_{b_h,β_h}	Flag of platform change of train h. $x_{b_h,\beta_h} = 1$, if $b_h = \beta_h$; $x_{b_h,\beta_h} = 0$, if $b_h \neq \beta_h$.
$\mathcal{R}^p_h$	Set of optional receiving routes of train h, $\mathcal{R}^p_h \subset \mathcal{R}^p$.	t_i	The decision moment of the i-th step.
$\mathcal{R}^q_h$	Set of optional departure routes of train h, $\mathcal{R}^q_h \subset \mathcal{R}^q$.	s_i	State at the i-th step.
$\mathcal{U}$	Set of entry points of the railway station, indexed by u.	h_i	Decision object train under state s_i.
$\mathcal{V}$	Set of exit points of the railway station, indexed by v.	A_i	Masked action space under state s_i.
$\mathcal{B}_h$	Set of optional berthing tracks of train h.	a_i	Selected action under state s_i.
t^a_h	Scheduled arrival time of train h.	μ_i	Reward under state s_i.
t^d_h	Scheduled departure time of train h.	M_i	Set of succeeding tuples of decision moment and decision object train under state s_i.
t^e_h	The earliest receiving time of train h.	a^r	Dispatching action associated with route r.
$\hat{t}^d_h$	Updated scheduled departure time of train h after train h received in the station.	a^ω	Postponing action.
b_h	Scheduled berthing track of train h.	λ_r	The next available time of route r.
b^r	Berthing track connected by route r	η	Clearance buffer between two consecutive trains assigned to the same platform.
Δ_h	Primary delay of train h.	τ^{a^r}	The time of claiming route r by action a^r.
τ^a_h	Actual arrival time of train h.	$\tau^{a^\omega}_h$	The time of selecting the postponing action for train h
τ^d_h	Actual departure time of train h.	t^m_h	The time of train h reaching the waiting time limit.
τ^e_h	Actual receiving time of train h.	φ	Waiting time limit for a train when arrive or depart.
τ^z_h	Actual leaving time of train h.		
β_h	Actual assigned berthing track of train h.		

Appendix B. Action mask

Under state s_i at the decision moment t_i, the masked action space A_i can be determined according to the decision moment category and the decision object train h_i, which is formulated as

$$A_i = \begin{cases} \{a^r \mid \lambda_r \le t^a_{h_i} - y_r, r \in \mathcal{R}^p_{h_i}\} \cup \{a^\omega\}, & (t_i, h_i) \in M^1_i, \text{ i.e., } t_i = t^e_{h_i} \quad \text{(B − 1a)} \\ \{a^r \mid \lambda_r \le t_i, \beta_{h_i} = b^r, r \in \mathcal{R}^q_{h_i}\} \cup \{a^\omega\}, & (t_i, h_i) \in M^2_i, \text{ i.e., } t_i = \hat{t}^d_{h_i} \quad \text{(B − 1b)} \\ \{a^r \mid \lambda_r \le t_i, r \in \mathcal{R}_{(h_i,t_i)}\}, & (t_i, h_i) \in M^3_i, \text{ i.e., } t_i = t^m_{h_i} \quad \text{(B − 1c)} \\ \{a^r \mid r = \text{domain}(\lambda_r)\} \cup \{a_\omega \mid t_i - \tau^{a^\omega}_{h_i} < \varphi\}, & (t_i, h_i) \in M^4_i, \text{ i.e., } t_i = \lambda_r \quad \text{(B − 1d)} \end{cases} \tag{B.1}$$

where $\mathcal{R}^p_{h_i}$ and $\mathcal{R}^q_{h_i}$ are the optional receiving and departure routes of train h_i, respectively; β_{h_i} is the berthing track where train h_i is assigned to; b^r is the berthing track that route r connects; $\tau^{a_\omega}_{h_i}$ is time of selecting the previous postponing action for train h_i, as we defined in Eq. (9). We elaborate on the masked action space in detail as follows.

Eq. (B.1a) means decision moment t_i of state s_i is the earliest receiving time of train h_i. In this case, the dispatching action with available receiving routes and the postponing action constitute the masked action space.

The condition to identify an available receiving route r is that train h_i can arrive on time using route r, i.e., the next available time of route (λ_r) is no later than the scheduled arrival time of train h_i minus the travel time of route r. If a dispatching action is selected from this masked action space, train h_i will arrive at the station on time. Therefore, its actual arrival time $\tau^a_{h_i}$ is the same as $t^a_{h_i}$, and its actual receiving time $\tau^e_{h_i}$ equals to $t^a_h - y_r$, where r is the selected receiving route.

Eq. (B.1b) means decision moment t_i is the updated scheduled departure time of train h_i ($\hat{t}^d_{h_i}$). As we stated in Section 3.1, only the departure route connected to β_{h_i} can be the dispatching action to trigger train h_i to depart. Hence, in this case, the condition to identify this departure route is $\beta_{h_i} = b^r$, and its availability is determined by comparing its next available route and the current time. If this dispatching action is selected, $\hat{t}^d_{h_i}$ is adopted as the actual departure time of train h_i ($\tau^d_{h_i}$), and the actual leaving time ($\tau^z_{h_i}$) is also determined according to the route travel time.

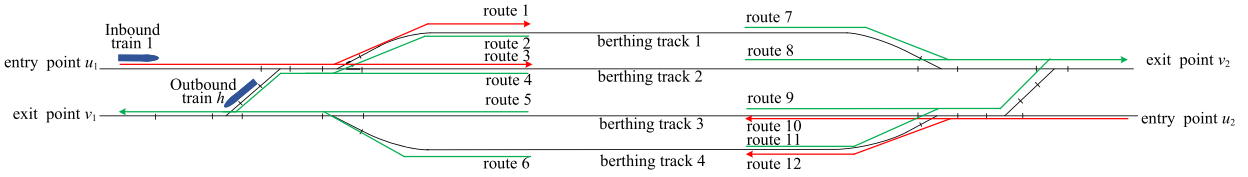
Eq. (B.1c) indicates state s_i locates at the waiting time limit of postponed train h_i (i.e., $t^m_{h_i}$). The masked action space A_i includes dispatching actions associated with current available routes of train h_i. The set of current available routes, denoted as set $\mathcal{R}_{(h_i,t_i)}$, depends on whether train h_i is an inbound train or an outbound train, which can be expressed as

$$\mathcal{R}_{(h_i,t_i)} = \begin{cases} \{r \mid \lambda_r \le t_i, \forall r \in \mathcal{R}^p_{h_i}\}, & \text{if } h_i \text{ is an inbound train} \quad \text{(B − 2a)} \\ \{r \mid \lambda_r \le t_i, \beta_{h_i} = b^r, \forall r \in \mathcal{R}^q_{h_i}\}, & \text{if } h_i \text{ is an outbound train} \quad \text{(B − 2b)} \end{cases} \tag{B.2}$$

That is, the next available time of route is the standard condition to identify available routes under current time t_i. The additional connection condition for outbound trains is identical to Eq. (B.1b). If a dispatching action associated with a receiving(departure) route is selected for train h_i in this situation, then the current decision moment t_i is adopted as the actual receiving (departure) time of train h_i, and the actual arrival (leaving) time is also determined according to the route travel time simultaneously. We note that the postponing action is not included in this case, since train h_i has already reached the waiting time limit. If no available route can be selected, then this state will be skipped directly.

Eq. (B.1d) indicates state s_i locates at the next available time of route r (i.e., $t_i = \lambda_r$). Hence, the availability of route r is guaranteed, and the masked action space A_i contains the unique dispatching action a_r. The postponing action is also added if train h_i has not reached the waiting time limit. If a dispatching action is selected, the actual receiving/arrival/departure/leaving times of train h_i are determined according to the current time t_i, same as we stated in the above paragraph for Eq. (B.1c).

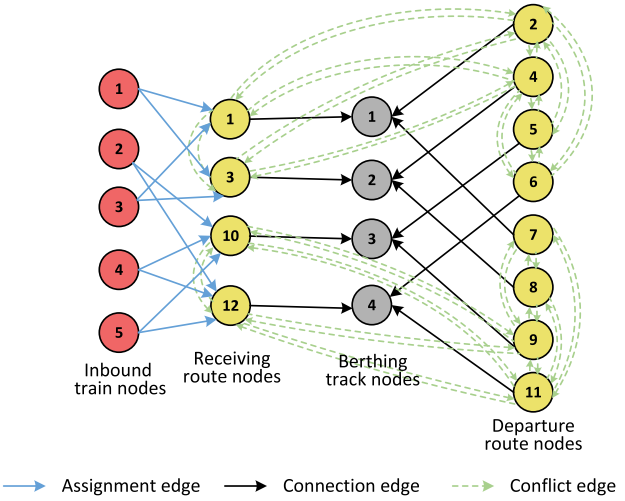
Appendix C. An illustrative example



(a) The yard layout of the illustrative example, and a snapshot of train operation simulation at 10:03:00

ID	Scheduled arrival time	Scheduled departure time	Entry point	Exit point	Scheduled platform
1	10:02:00	10:06:00	u_1	v_2	1
2	10:08:00	10:28:00	u_2	v_2	4
3	10:16:00	10:20:00	u_1	v_2	2
4	10:32:00	10:35:00	u_2	v_1	3
5	10:40:00	10:43:00	u_2	v_1	3

(b) The timetable of next 5 inbound trains



(c) The graph structure representing the state s_{i+1}

Fig. C.1. An illustrative example, including (a)the yard layout, (b) the timetable, and (c) the graph encoded state.

To further aid the understanding of the core design in our L2RP framework: the state transition and the graph encoded state, we visualize an illustrative example with a small station, which has 2 entry points, 2 exit points, 4 berthing tracks, and 12 routes, as shown in Fig. C.1. The 4 receiving routes are represented in red lines, and 8 departure routes are represented in green lines in Fig. C.1(a).

In this example, we set the clock of the current state s_i to 10:02:00, i.e., the scheduled arrival time of the delayed inbound train 1. The predecessor state s_{i-1} is located at 10:01, where the outbound train h begins to depart via route 4. Since the DOC value between route 4 and route 1 is 120 seconds, the *next available time* of route 1 is set to 10:03:00. Therefore, at state s_i, inbound train 1 cannot be received to arrive due to route conflicts and is postponed to arrive via route 1. Meanwhile, according to the method of determining the succeeding decision moment, which is introduced in Section 4.1.3, at state s_{i-1}, a succeeding decision moment is set at 10:03:00, i.e., the *next available time* of route 1. Hence, state s_{i+1} locates at 10:03:00, where route 1 becomes available, and inbound train 1 will use this route if the DRL agent takes the dispatching action associated with route 1.

The DRL agent makes decisions relying on the graph encoded state, which is introduced in Section 4.2. Under state s_{i+1}, the graph representing the state is completely constructed as Fig. C.1(c). In this graph, we include the following 5 inbound trains scheduled in the given timetable. The conflict edges among all 12 routes are established. For example, the receiving route 10 has shared sections with routes 9, 11, and 12, thus two directed conflict edges are established between each pair of route nodes. The assignment and

connection edges are established according to inbound trains' optional platforms and route accessibility, respectively. Then, the DRL agent will choose an action for state s_{i+1} based on learning over this graph. If it chooses route 1, train 1 will arrive at berthing track 1.

Appendix D. Schematic figures of yard layouts

The schematic figures of the two real-world railway yard layouts stated in Section 5.1 are as follows.

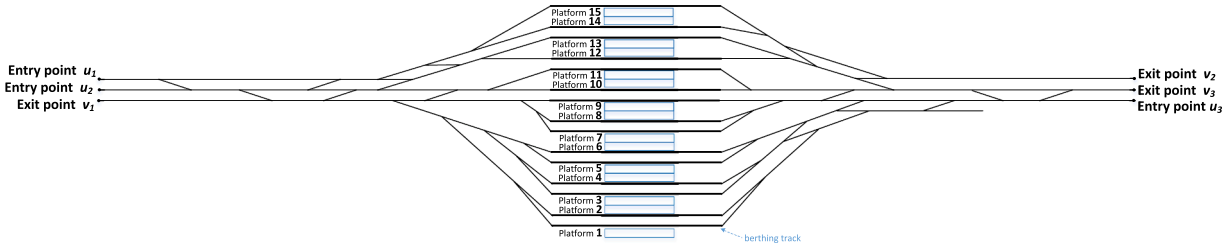


Fig. D.1. The schematic layout of ZhengXi yard in Xi'an North high-speed railway station.

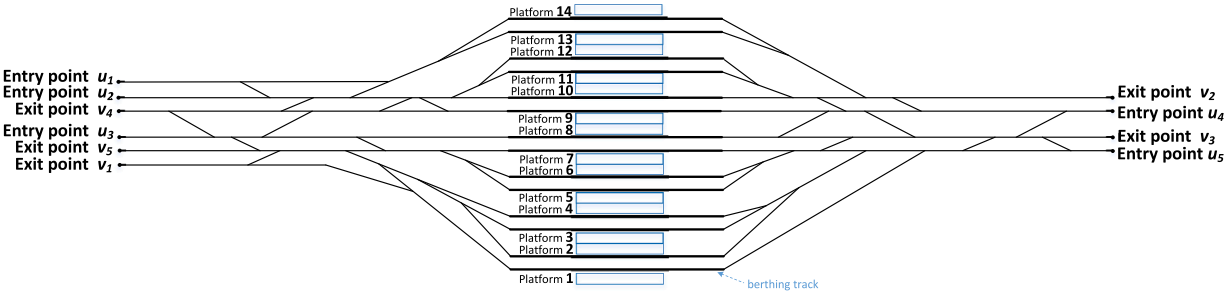


Fig. D.2. The schematic layout of JingGuang yard in Zhengzhou East high-speed railway station.

References

Bettinelli, A., Santini, A., Vigo, D., 2017. A real-time conflict solution algorithm for the train rescheduling problem. Transport. Res. Part B: Methodological 106, 237–265. https://doi.org/10.1016/j.trb.2017.10.005

Bigi, F., Bosi, T., Pineda-Jaramillo, J., Viti, F., D'Ariano, A., 2024. Long-term fleet management for freight trains: assessing the impact of wagon maintenance through simulation of shunting policies. J. Rail Transport Plann. Manage. 29, 100430.

Billionnet, A., 2003. Using integer programming to solve the train-platforming problem. Transport. Sci. 37 (2), 213–222.

Bosi, T., Bigi, F., D'Ariano, A., Viti, F., Pineda-Jaramillo, J., 2024. Optimal management of full train load services in the shunting yard: a comprehensive study on shunt-in shunt-out policies. Comput. Ind. Eng. 188, 109865.

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

Cadarso, L., Maróti, G., Marin, A., 2015. Smooth and controlled recovery planning of disruptions in rapid transit networks. IEEE Trans. Intell. Transp. Syst. 16 (4), 2192–2202.

Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P., 2023. Combinatorial optimization and reasoning with graph neural networks. J. Mach. Learn. Res. 24 (130), 1–61.

Caprara, A., Galli, L., Toth, P., 2011. Solution of the train platforming problem. Transport. Sci. 45 (2), 246–257. Publisher: INFORMS. https://doi.org/10.1287/trsc.1100.0366

Carey, M., Carville, S., 2003. Scheduling and platforming trains at busy complex stations. Transport. Res. Part A: Policy Pract. 37 (3), 195–224. https://doi.org/10.1016/S0965-8564(02)00012-5

Chakroborty, P., Vikram, D., 2008. Optimum assignment of trains to platforms under partial schedule compliance. Transport. Res. Part B: Methodological 42 (2), 169–184. https://doi.org/10.1016/j.trb.2007.07.003

Corman, F., Goverde, R. M.P., D'Ariano, A., 2009. Rescheduling dense train traffic over complex station interlocking areas. Robust and online large-scale optimization: models and techniques for transportation systems , 369–386.

Dewilde, T., Sels, P., Cattrysse, D., Vansteenwegen, P., 2013. Robust railway station planning: an interaction between routing, timetabling and platforming. J. Rail Transport Plann. Manage. 3 (3), 68–77. https://doi.org/10.1016/j.jrtpm.2013.11.002

García-Ródenas, R., López-García, M.L., Cadarso, L., Codina, E., 2024. An efficient greedy heuristic for the real-time train platforming problem. Comput. Oper. Res. 164, 106525. https://doi.org/10.1016/j.cor.2023.106525

Jusup, M., Trivella, A., Corman, F., 2021. A review of real-time railway and metro rescheduling models using learning algorithms. In: 21St Swiss Transport Research Conference (STRC 2021). STRC.

Kang, L., Lu, Z., Meng, Q., 2019. Stochastic schedule–based optimization model for track allocations in large railway stations. J. Transport. Eng., Part A: Syst. 145 (3), 04019001.

Khadilkar, H., 2018. A scalable reinforcement learning algorithm for scheduling railway lines. IEEE Trans. Intell. Transp. Syst. 20 (2), 727–736.

Lamorgese, L., Mannino, C., 2015. An exact decomposition approach for the real-time train dispatching problem. Oper. Res. 63 (1), 48–64.

Li, W., Ni, S., 2022. Train timetabling with the general learning environment and multi-agent deep reinforcement learning. Transport. Res. Part B: Methodological 157, 230–251.
Lin, Z., Wu, Y., Zhou, B., Cao, Z., Song, W., Zhang, Y., Jayavelu, S., 2024. Cross-problem learning for solving vehicle routing problems. arXiv preprint arXiv:2404.11677 .
Lu, G., Ning, J., Liu, X., Nie, Y.M., 2022. Train platforming and rescheduling with flexible interlocking mechanisms: an aggregate approach. Transport. Res. Part E: Logistics and Transportation Review 159, 102622. https://doi.org/10.1016/j.tre.2022.102622
Lu, G., Zhang, H., Shen, Z., Liu, X., 2025. Refining arrival headway for high-speed trains approaching a large railway station: a speed profile intervention approach. Railway Eng. Sci. 33 (3), 496–520.
Luangboriboon, N., Samà, M., D'Ariano, A., Fujiyama, T., 2025. Train platforming problem from the viewpoint of passenger flow management. Transportation , 1–23.
Lusby, R.M., Larsen, J., Ehrgott, M., Ryan, D., 2011. Railway track allocation: models and methods. OR Spectrum 33 (4), 843–883. https://doi.org/10.1007/s00291-009-0189-0
Meng, X., Wang, Y., Xiang, W., Jia, L., 2021. An integrated model for train rescheduling and station track assignment. IET Intel. Transport Syst. 15 (1), 17–30. https://doi.org/10.1049/itr2.12001
Narayanaswami, S., Rangaraj, N., 2013. Modelling disruptions and resolving conflicts optimally in a railway schedule. Comput. Ind. Eng. 64 (1), 469–481.
Pellegrini, P., Marlière, G., Rodriguez, J., 2014. Optimal train routing and scheduling for managing traffic perturbations in complex junctions. Transport. Res. Part B: Methodological 59, 58–80.
Pineda-Jaramillo, J., Bigi, F., Bosi, T., Viti, F., D'ariano, A., 2023. Short-term arrival delay time prediction in freight rail operations using data-driven models. IEEE Access 11, 46966–46978.
Reda, D., Tao, T., van de Panne, M., 2020. Learning to locomote: understanding how environment design matters for deep reinforcement learning. In: Proceedings of the 13Th ACM SIGGRAPH Conference on Motion, Interaction and Games, pp. 1–10.
Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
Sels, P., Vansteenwegen, P., Dewilde, T., Cattrysse, D., Waquet, B., Joubert, A., 2014. The train platforming problem: the infrastructure management company perspective. Transport. Res. Part B: Methodological 61, 55–72. https://doi.org/10.1016/j.trb.2014.01.004
Šemrov, D., Marsetič, R., Žura, M., Todorovski, L., Srdic, A., 2016. Reinforcement learning approach for train rescheduling on a single-track railway. Transport. Res. Part B: Methodological 86, 250–267.
Smit, I.G., Zhou, J., Reijnen, R., Wu, Y., Chen, J., Zhang, C., Bukhsh, Z., Nuijten, W., Zhang, Y., 2024. Graph neural networks for job shop scheduling problems: A survey. arXiv preprint arXiv:2406.14096.
Teng, J., Gao, J., Wang, P., Qu, S., 2023. Train rescheduling and platforming in large high-speed railway stations. Int. J. Transp. Sci. Technol. , S2046043023000989https://doi.org/10.1016/j.ijtst.2023.11.001
Wang, C., Cao, Z., Wu, Y., Teng, L., Wu, G., 2024. Deep reinforcement learning for solving vehicle routing problems with backhauls. IEEE Trans. Neural Netw. Learn. Syst.
Wu, Y., Song, W., Cao, Z., Zhang, J., Lim, A., 2021. Learning improvement heuristics for solving routing problems. IEEE Trans. Neural Netw. Learn. Syst. 33 (9), 5057–5069.
Yang, F., Yang, Y., Ni, S., Liu, S., Xu, C., Chen, D., Zhang, Q., 2023. Single-track railway scheduling with a novel gridworld model and scalable deep reinforcement learning. Transport. Res. Part C: Emerg. Technol. 154, 104237. https://doi.org/10.1016/j.trc.2023.104237
Ye, Y., Zhang, X., Sun, J., 2019. Automated vehicle's behavior decision making using deep reinforcement learning and high-fidelity simulation environment. Transport. Res. Part C: Emerg. Technol. 107, 155–170.
Ying, C., Chow, A. H.F., Chin, K.-S., 2020. An actor-critic deep reinforcement learning approach for metro train scheduling with rolling stock circulation under stochastic demand. Transport. Res. Part B: Methodological 140, 210–235.
Ying, C.-S., Chow, A. H.F., Wang, Y.-H., Chin, K.-S., 2022. Adaptive metro service schedule and train composition with a proximal policy optimization approach based on deep reinforcement learning. IEEE Trans. Intell. Transp. Syst. 23 (7), 6895–6906.
Yuan, J., Hansen, I.A., 2007. Optimizing capacity utilization of stations by estimating knock-on train delays. Transport. Res. Part B: Methodological 41 (2), 202–217.
Zhan, S., Kroon, L.G., Veelenturf, L.P., Wagenaar, J.C., 2015. Real-time high-speed train rescheduling in case of a complete blockage. Transport. Res. Part B: Methodological 78, 182–201. https://doi.org/10.1016/j.trb.2015.04.001
Zhang, B., Zhang, Y., D'Ariano, A., Bosi, T., Lu, G., Peng, Q., 2023a. Optimal platforming, routing, and scheduling of trains and locomotives in a rail passenger station yard. Transport. Res. Part C: Emerg. Technol. 152, 104160. https://doi.org/10.1016/j.trc.2023.104160
Zhang, C., Gao, Y., Cacchiani, V., Yang, L., Gao, Z., 2023b. Train rescheduling for large-scale disruptions in a large-scale railway network. Transport. Res. Part B: Methodological 174, 102786.
Zhang, H., Guo, M., Hu, L., Lu, G., 2024. A multi-agent simulation based train platforming research for facilitating passenger transfer in a high-speed railway station. Simul. Modell. Pract. Theory 130, 102856. https://doi.org/10.1016/j.simpat.2023.102856
Zhang, H., Lu, G., Zhang, Y., D'Ariano, A., Wu, Y., 2025a. Railcar itinerary optimization in railway marshalling yards: a graph neural network based deep reinforcement learning method. Transport. Res. Part C: Emerg. Technol. 171, 104970.
Zhang, J., Zhang, J., 2023. Artificial intelligence applied on traffic planning and management for rail transport: a review and perspective. Discrete Dyn. Nat. Soc. 2023 (1), 1832501.
Zhang, Q., Lusby, R.M., Shang, P., Liu, C., Liu, W., 2025b. Solving a multi-resolution model of the train platforming problem using lagrangian relaxation with dynamic multiplier aggregation. Eur. J. Oper. Res.
Zhang, Q., Lusby, R.M., Shang, P., Zhu, X., 2020a. Simultaneously re-optimizing timetables and platform schedules under planned track maintenance for a high-speed railway network. Transport. Res. Part C: Emerg. Technol. 121, 102823. https://doi.org/10.1016/j.trc.2020.102823
Zhang, Q., Zhu, X., Wang, L., 2018. Track allocation optimization in high-speed railway stations from infrastructure management and service perspectives. Meas. Control 51 (7–8), 243–259. https://doi.org/10.1177/0020294018785187
Zhang, Y., Zhong, Q., Yin, Y., Yan, X., Peng, Q., 2020b. A fast approach for reoptimization of railway train platforming in case of train delays. J. Adv. Transport. 2020, 1–20. https://doi.org/10.1155/2020/5609524
Zhu, Y., Goverde, R. M.P., 2020. Dynamic and robust timetable rescheduling for uncertain railway disruptions. J. Rail Transport Plann. Manage. 15, 100196.
Zhu, Y., Wang, H., Goverde, R. M.P., 2020. Reinforcement learning in railway timetable rescheduling. In: 2020Del InsThinspace IEEE 23Rd International Conference on Intelligent Transportation Systems (ITSC). IEEE, pp. 1–6.
Zhu, Y., Wang, P., Corman, F., 2023. Reinforcement learning in railway delay management. Available at SSRN 4436899.
Zwaneveld, P.J., Kroon, L.G., Romeijn, H.E., Salomon, M., Dauzere-Peres, S., Van Hoesel, S. P.M., Ambergen, H.W., 1996. Routing trains through railway stations: model formulation and algorithms. Transport. Sci. 30 (3), 181–194.
Zwaneveld, P.J., Kroon, L.G., Van Hoesel, S. P.M., 2001. Routing trains through a railway station based on a node packing model. Eur. J. Oper. Res. 128 (1), 14–33.