Solving the Train Dispatching problem via Deep Reinforcement Learning

Valerio Agasucci[a,b], Giorgio Grani[c,*], Leonardo Lamorgese[b]

[a] *DIAG, Sapienza University of Rome, Rome, Italy*
[b] *OPTRAIL, Rome, Italy*
[c] *Department of Statistical Sciences, Sapienza University of Rome, Rome, Italy*

Abstract

Every day, railways experience disturbances and disruptions, both on the network and the fleet side, that affect the stability of rail traffic. Induced delays propagate through the network, which leads to a mismatch in demand and offer for goods and passengers, and, in turn, to a loss in service quality. In these cases, it is the duty of human traffic controllers, the so-called dispatchers, to do their best to minimize the impact on traffic. However, dispatchers inevitably have a limited depth of perception of the knock-on effect of their decisions, particularly how they affect areas of the network that are outside their direct control. In recent years, much work in Decision Science has been devoted to developing methods to solve the problem automatically and support the dispatchers in this challenging task. This paper investigates Machine Learning-based methods for tackling this problem, proposing two different Deep Q-Learning methods(Decentralized and Centralized). Numerical results show the superiority of these techniques respect to the classical linear Q-Learning based on matrices. Moreover the Centralized approach is compared with a MILP formulation showing interesting results. The experiments are inspired on data provided by a U.S. class 1 railroad.

Keywords: Scheduling; Reinforcement Learning; Optimization

Declarations

- **Declaration of interests**

  Declaration of interests: none.

- **Funding**

*Corresponding author
Email addresses: agasucci@diag.uniroma1.it (Valerio Agasucci), g.grani@uniroma1.it (Giorgio Grani), leonardo.lamorgese@optrail.com (Leonardo Lamorgese)

The research conducted by Giorgio Grani and presented in this paper was partly funded by *Project Opstra* (nr.267554), supplied by *The Norwegian National Research Council*.

- **Authors' contribution**
  - **Valerio Agasucci :** Data curation; Formal analysis; Investigation; Methodology; Resources; Software; Validation; Visualization; Writing - original draft; Writing - review and editing.
  - **Giorgio Grani (corresponding author):** Conceptualization; Data curation; Formal analysis; Investigation; Methodology; Resources; Software; Validation; Visualization; Supervision; Writing - original draft; Writing - review and editing.
  - **Leonardo Lamorgese:** Resources; Writing - original draft; Writing - review and editing.

1. Introduction

A railway system is a complex network of interconnected tracks, where train traffic is controlled via the signaling system by activating switches and signals. The management and operation of a railway system is the task of the Infrastructure Manager (IM). A Train Operating Company (TOC) organizes its fleet to accommodate expected demands, maximizing revenue and coverage. In regions where for geographical and historical reasons the market is made up of predominantly freight-hauling companies, such as, e.g., North America and Australia, the IM and TOC are often the same entity. That is, rail operators are generally vertically integrated, owning and operating both the network and the fleet. In most countries however, the IM is a single public authority, which rents its network to TOCs to operate their train services. This is particularly common for systems that operate predominantly passenger traffic, like in Europe. Here, the IM interacts with the TOCs to establish a plan for train traffic, the so-called *timetable*. This process, referred to as timetabling, takes place offline, generally every 3 to 12 months, and is a challenging and time-consuming task. Its main goal is to assign routes to trains and create a schedule that is conflict-free and, typically, presents some elements of periodicity. Despite a vast body of literature in this field of research, in practice timetables are to this day still in many cases hand-engineered by specialised personnel, basing their decisions largely on experience, regulation, safety measures, business rules to factor in the various requests expressed by the TOC (the latter more delicate and time-consuming in liberalised markets with multiple competing TOCs).

In operating a railway system, the IM ideally attempts to adhere perfectly to the timetable. Unfortunately, as anyone who has ever taken a train will be familiar with, this seldom happens, as small disturbances, or in some cases serious disruptions, occur daily. A train malfunction, switch or track failure, delays in the preparation of the train or in passengers embarking, and plenty of other

issues may create train delays and ultimately affect the overall network, sometimes in unforeseen ways. In some cases small delays are recovered from simply driving trains faster, but, very often, online re-routing and re-scheduling decisions have to be taken to reduce delays and increase efficiency. In literature, this online decision making process is referred to as the Train Dispatching problem (`TD`), a real-time variant of the above mentioned Train Timetabling problem (known to be NP-hard [8]). The very little time admitted for computation (often only a few seconds) and the size and complexity of real-life instances makes this problem very challenging to solve also in practice. The real-time nature of the problem in particular largely limits the solution approaches that can be used effectively. Operationally, the task of dispatching trains is in the hands of the IM's traffic controllers, the *dispatchers*, each assigned to a specific portion of the network (a station, a junction, a part of a line, etc). To this day, dispatchers are generally provided with little or no decision support in this process, which makes it challenging for them to go beyond their local view of the network and take into account the knock-on effects of their decisions, especially in areas of the network that are outside their direct control.

The Train Dispatching problem has sparked much research interest over the years, in particular in the Optimization community. Different models and solutions approaches have been proposed over the years: some exact methods ([19]-[23]), heuristic based on MILP formulation as [42]-[4]-[5]-[1], tabu search [37], genetic algorithms [16], classic greedy heuristic [7], and neighbourhood search ([27]). Refer to [20] for a deeper insight on the Train Dispatching problem, and to the many surveys on the topic for an overview of these approaches (e.g. [6]-[11]-[14]- [22]-[36]). Recent developments in both Machine Learning and Optimization have led to the definition of new learning-based paradigms to solve hard problems. Our focus is on Reinforcement Learning, for which a lot of interesting results have been achieved so far. In particular, in the well known AlphaGo algorithm (see [34]) the authors tackle the complex game of Go, developing an outstanding framework able to overcome the best human player. Several approaches have followed AlphaGo, like AlphaZero [33, 32] and MuZero [29], increasing every time the degree of generalization possible. Recently in [18], this approach was extended to combinatorial problems with a more general range of possible applications. To achieve their results, the authors combined Deep Q-Learning with graph convolutional neural networks, which are a specialized class of models for graph-like structures. A very similar approach was followed in [13].

Our contribution. In this paper, `TD` is tackled by means of Deep Q-Learning on a single railway line. More specifically, two approaches are investigated: Decentralized and Centralized. In the former, each train can be seen as an independent agent with the ability to see only a part of the network, namely some tracks ahead/behind it and not beyond. This approach has the advantage that it can be easily generalized, but, on the other hand, may lack the depth of prediction useful to express network dynamics. The latter method takes as input the entire line and learns to deal with delay propagation. Moreover, we

use a Graph Neural Network to estimate this delay reflecting the railway topology. Both methods generalize to different railway sizes. From the standpoint of the application, these authors find the use of Deep Reinforcement Learning approaches (such as Deep Q-Learning) very promising, as they present the advantage of shifting the computational burden to the learning stage. While enumerative algorithms typical of Combinatorial Optimization and Constraint Programming have proven to be effective in several train dispatching contexts, scalability remains a daunting challenge. On the other hand, under the assumption that the model can be trained effectively, Deep RL could achieve a (quasi) real-time performance, unattainable with classical optimization approaches. Finally, we point out that a possible reason for the slow adoption of automatic dispatching software in the industry is the diversity of business rules and requirements for such software in different regions and markets. The step of adapting the software to a specific set of such rules and requirements could be accelerated (or indeed skipped) using a Deep RL-based approach, which builds its internal input-output representation based on provided data and requires virtually no knowledge of the rules themselves.

For all the above, we believe that it is worth pursuing the application of Deep RL techniques in the field of train dispatching. This article is not the first article proposing the use of RL for train dispatching, so we highlight the differences between these. Papers [17]-[30] use a linear Q-Learning approach to tackle the problem. In [15], the authors present an approach based on approximate dynamic programming. Recent papers [24],[25],[39] introduce a Deep-network to predict the next action. In [21] Deep RL is used to solve the energy-aimed timetable rescheduling (find the optimal timetable minimizing the energy usage) in [41] an RL approach is exploited to reschedule the timetable of the high-speed railway line between Shangai and Beijing, while in [40] a multi-agent reinforcement learning is incorporated in the metro system. Finally in [44]-[45] an Actor-Critic method is used to schedule the underground train service in London. However, these approaches are quite limited in terms of the size of the instances that can be solved. Indeed, in [24],[25],[39] the railway network considered in the experiments is limited to 7 or 8 stations and in [24],[39] train traffic is considered only in one direction. In this paper we present an approach which is able to tackle larger instances (up to 29 stations in our experiments) and handle both traffic directions. Furthermore, we model other, important factors in the dispatching process, such as train length and other business rules described in section 2.1, which instead are not all covered in the cited papers. The train length has been tackled in other works among the ones cited before, but it appears to be a factor that increases significantly the computational complexity of the proposed procedures, where here is directly embedded. This results in a model that is more faithful to real-life requirements and an algorithm that can solve instances of some practical relevance (e.g. a regional line). In addition, it may be worth noting that our experiments are carried out on data from the US railway network, which presents its own challenges respect to other regions in the world like the European, Chinese and Japanese ones. Finally, in the computational section we show how the Deep Q-Learning approaches here

presented perform better than their linear counterpart, the matrix Q-learning approach proposed in [17].

The paper is organized as follows: in Section 2, basic concepts of Train Dispatching and Reinforcement Learning are introduced formally. In Section 3 Graph Neural Networks are introduced. In Section 4, the two algorithms are discussed. Specifically, in sub-section 4.1, the Decentralized approach is tackled, whereas in sub-section 4.2, the Centralized approach is presented. In Section 5, a numerical analysis is conducted to prove the effectiveness of Deep Reinforcement Learning approach for the train dispatching problem. Finally in Section 6 a comparison between our approach and a well-known Mixed-Integer Programming approach is provided.

2. Preliminaries

Two basic ingredients characterize this paper: the Train Dispatching problem (`TD`) and (Deep) Reinforcement Learning (`DeepRL`). In the following, a short presentation of both, with the aim of being introductory and not comprehensive.

2.1. Train Dispatching problem

The fundamental elements of the Train Dispatching problem (`TD`) are trains, the railway network, and paths. Trains are of course the means of transportation in this system. In practice, different types of trains can operate at the same time on a network. Two of the most important attributes of a train for dispatching decisions are its priority and its length. Priority represents the relative importance of a train respect to other trains, which generally reflects in the decisions taken by dispatchers to recover from delay. For example, cargo trains usually have a lower priority than passenger ones. Their length depends on various aspects, such as technology, market and nature of the service.

A railway network is composed of a set of tracks, switches and signals. A common way to represent this network is to see it as an alternating sequence of tracks and stations. A station is a logical entity in the network that comprises multiple tracks and switches where, typically, the majority of routing and scheduling actions take place. Tracks are physical connections between two stations. Trains traveling in opposite directions can never occupy a track simultaneously, while certain tracks allow this for trains travelling in the same direction.

In terms of infrastructure, stations are effectively also a set of interconnected track segments. An interlocking route is a sequence of track segments that connects two signals. Different routes can be conflicting, that is, certain movements may not be allowed on such routes at the same time. The interlocking is precisely the signaling device that prevents these conflicts. Certain interlocking routes also include stopping points, which allow train activities. Some of these activities can be embarking passengers or loading goods. Generally each stopping point can be occupied by at most one train at the time. A switch is

a mechanical installation enabling trains to be guided from one track to the other. A switch can be occupied by one train at a time. Finally, a path is the expected sequence of tracks and stations for a given train. The path is specified by a starting station, a set of boarding points, and the arrival station.

The elements introduced above allow to model real-life operations with a level of approximation that is sufficient for the purposes of this paper.

In a few words, `TD` is the problem of managing train traffic in real-time by taking scheduling and routing decisions in order to maximize system efficiency. Such dispatching decisions are subject to a number of constraints related to the signaling system, infrastructure capacity, business rules, train physics etc. In railway systems operated following an official timetable, the goal is typically to adhere to it as much as possible. When deviations from this plan occur, the dispatcher objective is to recover from these deviations, minimizing train delay.

A particularly pernicious situation is the occurrence of a *deadlock*. A group of trains are said to be in deadlock if none of them can move because of another train that is blocking the next track of its path. A deadlock is the result of lack of information (e.g. wrong train length assumptions) or, more often, induced by human error (erroneous dispatching decisions). A critical aspect of any automatic dispatching systems is that it avoid creating deadlocks at all costs.

2.2. Reinforcement Learning

Reinforcement Learning (`RL`) is an approximated version of Dynamic Programming, as stated in [3]. This paradigm learns, in the sense that the approximated function built takes into account statistical information obtainable from previous iterations of the same algorithm or external data. The term Deep Reinforcement Learning (`DeepRL`) usually refers to `RL` where a deep neural network is used to build the approximation. We refer to [35] and [3] for a comprehensive discussion on `RL` .

`RL` is usually explained through agent and environment interaction. The agent is the part of the algorithm that learns and takes decisions. To do so, it analyzes the surrounding environment. Once it takes a decision, the environment reacts to this decision and the agent perceives the effect its action has produced. To understand if the action taken has been successful or not, the agent may receive a reward associated with the action and the new environment observed.

In the Decentralized approach in Section 4.1, the agent is the train and the environment the observable line , i.e. the portion of line and trains reachable by the agent, more details in Section 4.1, as opposed to the centralized approach, the agent act as a line coordinator, deciding for each train, and the environment is the entire line.

For the purposes of this paper, the reinforcement procedure will move forward following the rolling horizon of events, indexed by t. An event takes place every time there is a decision to make that may affect the global objective. In the `RL` vocabulary, the state represents the formal representation of the environment at a time step t, and it is formalized by the vector $s_t \in \mathcal{S}_t$, where

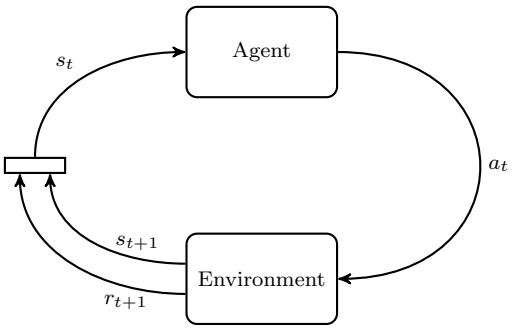

Figure 1: The reinforcement learning framework

$\mathcal{S}_t \subseteq \mathbb{R}^m$ is the set of all possible states at time t. The action taken by the agent is $a_t \in \mathcal{A}_t$, where $\mathcal{A}_t \subseteq \mathbb{R}^n$ is the set of all possible actions at time t. Once the agent observes s_t and carries out a_t, the environment reacts by producing the new state $s_{t+1} \in \mathcal{S}_t$ and a reward $r_{t+1} \in \mathcal{R}$, where $\mathcal{R} \subseteq \mathbb{R}$ is the set of all possible rewards. In this case, $\mathcal{R}$ is mono-dimensional, but in general, it could take the form of a vector, depending on the problem examined. Figure 1 shows a simplified flowchart of `RL` .

Q-learning is a branch of `RL` that uses an action-value function (usually referred to as q-function) to identify the action to take. More formally, the q-function $Q(s_t, a_t)$ represents the reward that the system is expected to achieve by taking said action given the state. At each step t, the action that brings the highest reward is chosen, namely:

$$a_t = \arg\max_{a \in \mathcal{A}_t} Q(s_t, a)$$

Pseudo-code for a Q-learning algorithm is presented in 1.

$\mathcal{L}(\hat{y}, y)$ is the loss function used to perform the training, i.e. a measure of the error committed by the model that one wants to minimize. Examples of loss functions are Mean Squared error, Cross-Entropy, Mean Absolute Error and so on, see [38] for an overview on the topic. $\epsilon \in (0, 1)$ is the probability of choosing a random action, T is the final state of the process (i.e. when it is not possible to move anymore), $\gamma \in (0, 1]$ is the future discount, a hyper-parameter reflecting the fact that future rewards may be less important. To make the estimation more consistent, usually, a replay mechanism is used, so that the agent interacts with the system for a few virtual steps before learning. This is suitable in situations where the environment can be efficiently manipulated or simulated in a way that the computational cost is only marginally affected.

3. Graph neural networks

Graph convolutional neural networks, introduced by [28], are used to approximate the Q-value function. Differently from Deep-neural network, GNN

Algorithm 1: Q-learning algorithm

Input: $\mathcal{P}$ a set of instances, $\mathcal{D} = \emptyset$ the memory, a loss function $\mathcal{L}(\hat{y}, y)$, $\epsilon \in (0, 1)$, $\gamma \in (0, 1]$, `#episodes`, `#moves`
Output: the trained predictor $Q(\cdot, \cdot)$
for $k = 1, \ldots,$ `#episodes` **do**
  Sample $P \in \mathcal{P}$
  Initialize episode $t = 0$, $s_0 = 0$
  for $t = 0, \ldots,$ `#moves` **do**
    Select an action $a_t = \begin{cases} Unif\{\mathcal{A}_t\}, & \text{with probability } \epsilon \\ \arg\max_{a \in \mathcal{A}_t} Q(s_t, a), & \text{with probability } 1 - \epsilon \end{cases}$
    Observe $s_{t+1}, r_{t+1} =$ `ENVIRONMENT`(s_t, a_t)
    Store (s_t, a_t, y_{t+1}) in the memory $\mathcal{D}$, where
    $y_{t+1} = \begin{cases} r_{t+1}, & \text{if } t + 1 = T \\ r_{t+1} + \gamma \arg\max_{a \in \mathcal{A}_t} Q(s_{t+1}, a), & \text{oth.} \end{cases}$
    Sample batch $(x, y) \subseteq \mathcal{D}$
    Learn by making one step of stochastic gradient descent w.r.t. the loss $\mathcal{L}(Q(x), y)$
    if $t + 1 = T$ **then**
      Break
    end
  end
end

exploits the graph input structure of the railway to learn the Q-value function. The name *convolutional* derives from the fact that the information is aggregated, shared and propagated among nodes according to how they are connected. This process is known as message passing, since the information of each node is propagated to the others for a given number of steps. The outputs of message passing are the so-called embedding vectors (one embedding for each node of the graph). The embedding vector collects information not only provided by the node to which it refers, but it brings information by other nodes of the graph. For richer overview of GNN, please refer to [46].

The input of the network is a graph $G(N, A)$ where N is the set of vertices and A the set of arcs. At each node $j \in N$ is associated a vector of features $x_j \in \mathbb{R}^n$ where n is the number of features. In our message passing architecture, we use the scheme adopted by [2]. Given an input graph, for each node i we define the set of neighbours $\delta(i)^- = \{j \in V : (j, i) \in E\}$.

The first step of the GNN is to apply a mean function to all the neighbour features of each node i:

$$m_i = \frac{\sum_{j \in ne_i} x_j}{|ne_i|}, \quad i \in V \tag{1}$$

Where m_i is the mean of the features of the neighbours of node i. In Figure 2 we show an example of a graph, the features and the mean (computed as reported above) associated to each node. Then a concatenation is performed between m_i and x_i, and put as input of a feed-forward neural network $ReLU1$) with $ReLU$ (see [31]) as an activation function:

$$l_i = ReLU(W_1^\intercal \, concat(x_i, m_i) + b_1), \quad i \in V$$

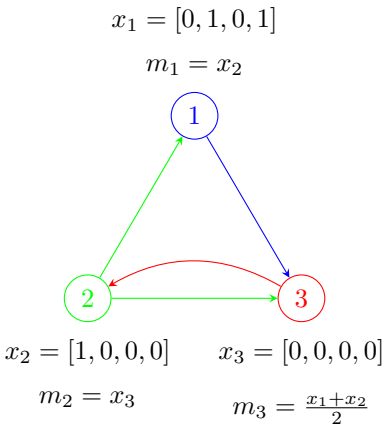

Figure 2: Graph and its features.

The term l_i is the output, W_1 the weights and b_1 the biases of the neural network $ReLU1$.

Depending on the node connections the outputs are passed to a second neural network $ReLU2$ with $ReLU$ as an activation function.

$$h_i = ReLU\left(W_2^\intercal\left(l_i + \sum_{j \in ne_i} l_j\right) + b_2\right), \quad i \in V$$

Where h_i is the embedding vector generated as the output of the neural network $ReLU2$, having W_2 and b_2 as weights and biases, respectively. The message passing returns an h_i for each $i \in N$. All the steps in the message passing can be performed several times, however for most applications 2 or 3 times are sufficient. In our experiments, we do so twice. The message passing ensures that the information of each node is propagated not only to its neighbours, but also to the furthest nodes. $ReLU1$ and $ReLU2$ have the same category of parameters: W_1 and b_1 for $ReLU1$; W_2 and $b2$ for $ReLU2$. The embedding vector h_i may have a different size than x_i. This is justified by the fact that h_i does not only bring the information from x_i but also from other nodes. To have another message passing step h_i becomes the new input of (1). Finally to obtain the target (in this work the Q-values) a last network $f(\cdot)$ is applied to the sum of h_i.

$$o = f\left((h_i)_{i \in V}\right)$$

In Figure 3, we show the network described above referred to graph in Figure 2.

4. Proposed methods

In this section, basic ideas regarding states, actions, and rewards to solve `TD` are presented. Two approaches are proposed: Decentralized and Centralized.

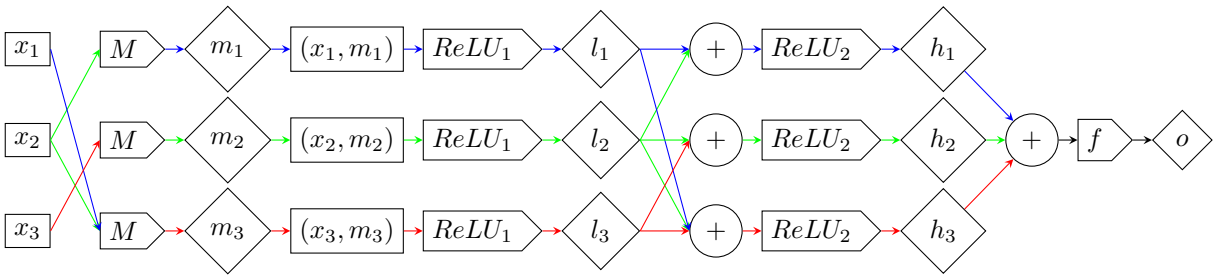

Figure 3: Message passing on the graph in figure 2.

The difference between them lies mainly in the topology of the state and in the reward mechanism, and therefore in the ability of the approximating q-function to capture the right policy. When describing the state, both of the approaches use the concept of resource. A resource is a track or a stopping point that can be occupied by one or more trains.

The majority of models for `TD` rely on several approximations to make the problem more readable and the mathematics easier to handle. Using `RL` allows to integrate many of these hidden aspects in the environment, introducing a new level of complexity with a relatively small effort. One of the most important is train length, which translate into computational burden for traditional optimization algorithms, where here is a feature directly embedded in the system. To avoid this, some models in the literature tend to approximate a train with a point, but this is not the case in real life where a train may occupy more than one resource at the same time. This is especially critical in railway systems that operate predominantly freight services (such as in the North American market), where trains are often longer than the available infrastructure to accommodate them, and the risk of dispatcher-induced deadlocks is very real.

Another important step is to model safety rules like the safety distance inside the same track, or the role of switches, For each track, we check the safety distance between two consecutive trains. Additionally, we model the rule that if a switch is occupied by a train, then no other train can use the same switch. Therefore, the train that has to cross will be held until the switch is freed. Finally, we introduce the minimum headway time between the occupation of a track by consecutive trains. Given a couple of follower trains, we compute the elapsed time since the first train has entered. If this quantity is smaller than a certain threshold, the headway time, the train has to wait. In our experiments, the headway time, the safety distance and all the other railway parameters are exogenous value inspired by the US class 1 Railroad we are working with. A simple rule has been implemented to avoid deadlocks between two crossing trains in specific circumstances.In short, given a generic train T_0 positioned in R_0 that wishes to occupy the resource R_1, then the method evaluates the position

of all visible trains T_i and the single-track resources[1] R_i, for $i = 1, \ldots, K$[2], converging to R_1. For each train T_i, the flow of resources $\mathcal{F}_i$ from R_i to R_1 is then computed. Finally, if there exists at least one $i \in \{1, \ldots, K\}$ such that a train T_i occupy a resource R_i in $\mathcal{F}_i$, then T_0 must wait to avoid deadlock.

The inclusion of all these aspects allows a greater adherence to the real-world problem than many optimization approaches attain, and the relative ease in doing so is, in our opinion, one of the advantages of a deep `RL` approach.

4.1. Decentralized approach

In this subsection, we will present the Decentralized approach, introducing the structure of the state when full observability of the rail network is guaranteed.

As studied in [9, 43], enriching the information available to the agent (so the state) affects the space of policies to be learned. For this reason, the following six features are associated to each resource:

1. **status**, which is a discrete value chosen among stopping point, track, blocked[3], and failure[4]
2. **number of trains**
3. **train priority** (if there is more than one train, the train with the highest priority is reported)
4. **direction**, which is a discrete value chosen among: follower, crossing, or empty[5]
5. **length check**, a Boolean identifying if the train length is less than or equal to the resource size
6. **number of parallel resources**[6] w.r.t. to the current one

For that which concerns the learning process the model that approximates the q-function is represented by a feed-forward deep neural network (`FNN`) with two fully connected hidden layers of 60 neurons each, and a third layer mapping into the space of the actions. The output of the third layer is then combined with a mask, disabling infeasible moves. Figure 4 shows the structure of the network. Firstly, the input (the state) passes through a layer of neurons, where each neuron is associated with a ReLU (see [31]) activation function. The output of the first layer goes to a second one with the same kind of activation functions. Then, everything is multiplied by the action mask, filtering allowed actions from prohibited ones. An action mask is a vector, whose components are equal to one if the correspondent action is allowed, zero otherwise.

[1] A single-track resource is a resource that does not have a parallel resource

[2] R_i is directly connected with R_{i+1} $\forall i = 1, \ldots K$

[3] Temporarily reserved by another train.

[4] The resource is out of service.

[5] A train T1 is a follower for a train T2 if they have the same direction, otherwise T1 is a crossing train for T2.

[6] Given a resource R_1 that connect two resources R_A and R_B, a resource R_2 is a parallel resource of R_1 if R_2 connects R_A and R_B.

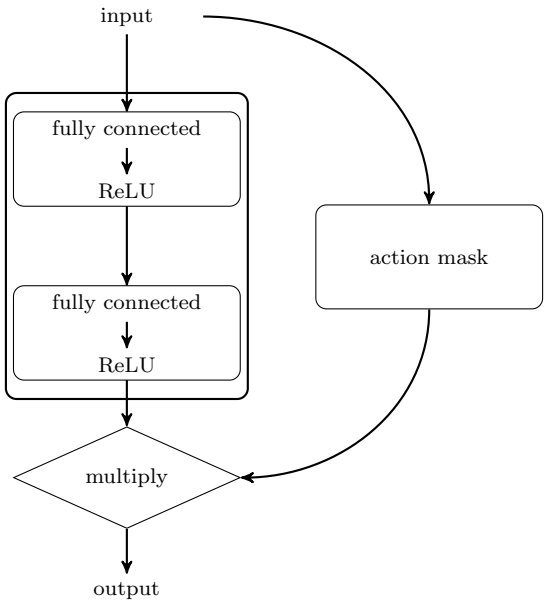

Figure 4: The deep network utilized

The typical goal of a dispatcher is to take routing and scheduling decisions that minimize some measures of delay. One way to model these decisions is to establish whether a train can access a specific resource or whether it has to stop/hold before entering it. The algorithm takes this decision when a train reaches a control point. When possible, the train will take the best-programmed resource as default. This happens when, for instance, all the resources ahead of the train are free. The best resource is the one ensuring a minimum programmed running-time for the specific train.

In Figure 5 , for example, we have four resources: 1a, 1b. 1c and 1d with respectively 800, 1000, 1200 and 1300 unimpeded running time, which is is the minimum running time that a train needs to go from its current position to its destination (in this example 1a, 1b, 1c and 1d) if it were never held.. The resource 1a represents the best choice, since it accumulates less unimpeded running time, while the alternatives (1b, 1c, 1d) are all higher.

On the other hand, it may happen that all the next reachable resources are not available, so the train is forced to stop. In all the other situations, the algorithm acts as if the train were able to take decisions by itself. The state of the system (from the point of view of the train) does not represent the entire network, but only a limited number of resources ahead and behind. For each visible resource, the six features discussed before are considered.

Given the state, the action to be taken is one of the following:

- halt

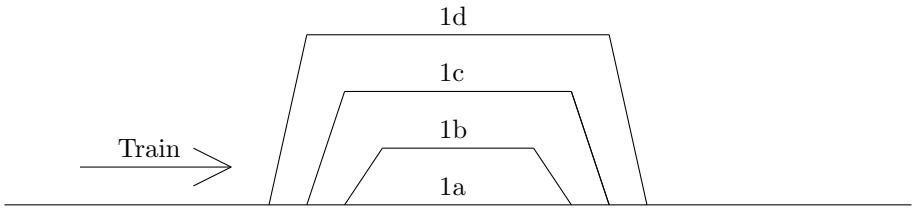

Figure 5: Best and reachable resources

- go to the best resource
- go to a reachable resource

The first action (stopping a train) can always be taken, while the other two possibilities depend on the state.

The most delicate part in the Decentralized approach is the definition of the reward. While the global objective is to minimize the total weighted delay, the agent viewpoint does not allow to express the reward associated with a state-action pair in terms of the actual impact on the overall network objective. For this reason, the reward is assigned at the end of each episode, and then the data collected is stored as $(s_t, a_t, Q(s_t, a_t), r_t)$, where $Q(s_t, a_t)$ is the output of the neural network. In particular, the reward for each generated state-action pair is given a large penalty value if the episode ends up in a deadlock, a minor penalty if the weighted delay is greater than 1.25 times the minimum weighted delay found so far, and a prize if the weighted delay is less than or equal to 1.25 times the minimum weighted delay found so far. This strategy is inspired by [17], where the authors adopt a classical matrix-based Q-learning approach on a single-track. We define the weighted delay of a train as the difference between the actual running time and the unimpeded running time. The actual running time is the timing of the solution taken by our algorithm, which may be different from the planned one, whereas the unimpeded running time is the the minimum running time that a train needs to go from its current position to its destination. By multiplying this value by a priority factor, leveraging how disruptive a delay would be, obtaining the cumulative delay of the train considered. The choice of leveraging reflects the different priorities of trains, so that a delay has more serious consequences if it refers to passenger train rather than a freight one. This is, in our experience, a common choice for companies. Summing up all the single delays we obtain the total weighted delay.

The last aspect to be discussed is memory management. Since no prior knowledge is used, the composition of the sample is critical to drive a smooth learning process. Therefore, the memory was divided into three data-sets: *best*, *normal* and *deadlock*. The *best* memory stores all the samples ending up in a weighted delay that is less than or equal to 1.25 times the best delay found so far, the *normal* memory stores the ones with weighted delay that is greater

than 1.25 the best found, and the *deadlock* memory stores all the unsuccessful instances.

Action-state-reward items are stored according to the level of reward if and only if the action mask in the `FNN` allows more than one action, so as to strengthen the learning process only on critical moves. Deadlock and normal memory data-sets are never deleted, while the best memory is updated every time a new best weighted delay is found.

4.2. Centralized approach

The idea behind the Centralized approach is that the `DeepRL` algorithm may take advantage of knowing the state of the overall network at each step, and attempt to learn the particular dynamics of the network thanks to the GNN. In this case the agent can be seen as a line coordinator, deciding critical issues at control points, predicting the expected effect on the network. This mimics to some extent the behaviour of human dispatchers, which have full control over a limited part of the network. Moreover, considering all the network as a state, the use of a GNN instead of a feed-forward neural network allows adopt the same neural network for different railway sizes. In feed-forward networks, the input is fixed in size, requiring a different model for every railway case, whereas using GNNs the same model operates to multiple railway networks.

In this approach the state is a graph, each node is a resource (track or stopping point) and there is an arc when two resources are linked. With each node is associated a vector of features that explains the characteristics of the train in that resource. This is a one hot-encoded vector with the following characteristics:

- the first n bits are reserved for the train's priority. The i-th bit is set to 1, if the train belongs to the i-th priority class, and 0 otherwise.
- one bit at position $n+1$, for the train's direction in that resource
- m bits, from position $n+2$ to position $n+2+m$, are reserved for the class length of the train
- finally, one bit in position $n+2+m+1$ states if a decision has to be taken for the train in that resource.

The action space is the same as the Decentralized approach, however the reward mechanism, unlike the Decentralized approach, takes into account the actual delay in absolute value, rather than a measure of how good such delay is with respect to the best achievable. More in detail we define the reward-to-go from a time step t as:

$$R_t = -\sum_{k=t}^{T} r_k$$

Where r_k is the difference of delay between step k and step $k-1$. The reward-to-go is a negative quantity because we are minimizing the delay. The Q-function,

instead, given a state s_t and the action a_t is given by:

$$Q(s,a) = \mathbb{E}[\sum_{k=t}^{T} r_k | s = s_t, a = a_t]$$

To estimate the reward-to-go and measure the goodness of an action, we have therefore to minimize the following loss:

$$\mathbb{L} = \sum_{t=1}^{T} (Q(s_t, a_t) - R_t)^2.$$

5. Numerical results

The test bed for the experiments is inspired on data provided by a U.S. class 1 railroad. The two algorithms were compared to the linear Q-learning algorithm proposed in [17], since the primary aim of this paper is to show that `DeepRL` is superior to linear Q-learning. In particular, the quality of the solutions in terms of final delay is the main driver used to discriminate the quality of an algorithm. For what concerns time, all the procedures need less than one second to complete an episode, and therefore are well suited for this online/real-time application.

All tests were performed on an Intel i5 processor, with no use of GPUs. Our experiments take into account railway networks with a different number of resources, number of trains and train's priorities.

The experiments were conducted to investigate the following properties:

1. the ability to solve unseen instances generated from the same distribution
2. the ability to generalize when the number and the length of the trains increase
3. the ability to generalize when the numbers of resources in the railway network changes.

5.1. First experiment

In this experiment the considered railway network is mainly characterized by the alternation of one station and one track (single-track), but also include some parts of the network where two stations are connected by two single tracks or more (multiple-track). In total, the network has 134 resources (tracks and stopping points), 15 tracks and 29 stopping points have at least one parallel resources, while 33 are single-track resources. The time window is two hours. In Figure 6, we report a small section of our railway to show what we consider as a stopping point and as a track. Resources 2a and 2b are stopping points where trains can meet or pass. Elements 1, 3, 4a, 4b, 5a and 5b are tracks, where the couples 4a, 4b and 5a, 5b are parallel, meaning they are connected to the same stations. We refer to both tracks and stopping points as resources. The track is effectively the portion of the network that connects two stations. Stations can

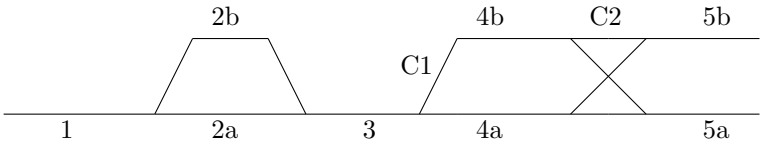

Figure 6: Railway example

have stopping points or they can be crossovers, like $C1$ or $C2$, where a crossover is composed by one or more switches connecting two parallel tracks, or a parallel track and a single one. We do not model the switches, considering them as a part of the next track. As explained in Section 4, if a train occupies a switch no other train can use it.

Traffic characteristics for each instance are described in terms of: number of trains, position, direction, priority and length.

The range for each parameter is realistic, as again it is inspired on input provided by the U.S. class 1 railroad. More specifically:

- the number of trains N is chosen randomly between 4 and 10 with the following probability distribution: $P(N = 4) = 0.1$, $P(N = 5) = 0.2$, $P(N = 6) = 0.2$, $P(N = 7) = 0.2$, $P(N = 8) = 0.15$, $P(N = 9) = 0.1$, $P(N = 10) = 0.05$
- the position is chosen using a uniform distribution on the available resources
- the direction is chosen uniformly
- the priority A is chosen randomly between 1 and 5 with the following probability distribution $P(A = 1) = 0.05$, $P(A = 2) = 0.15$, $P(A = 3) = 0.23$, $P(A = 4) = 0.27$, $P(A = 5) = 0.3$
- the length is chosen uniformly in the set {4000, 4500, 5000, 5500, 6000, 6500} to be intended in feet

Given the above mentioned distributions, to generate an instance we repeat the following steps:

1. select the number of trains N
2. for each train in N:
   (a) select its initial position
   (b) select its direction (down-hill, up-hill)
   (c) select its priority
   (d) select its length.

Priority, direction and position are sampled according to the distribution probability described above even in the next experiments. Additionally, the delay is multiplied by a penalty factor to express the relative importance of a train class and its priority. Given a priority $A \in \{1, 2, 3, 4, 5\}$, the coefficient ω_A

Table 1: Basic Statistics on 100 Test Instances From the Same Distribution of the Training.

Delay	Linear Q-learning	Decentralized	Centralized
minimum	886	0	0
average	42399.04	9914.864	9295.297
maximum	174262	30578	35441
std dev	35422.18	7287.825	7398.979
# deadlocks	4	5	4

is such that $\omega_1 = 20$, $\omega_2 = 10$, $\omega_3 = 5$, $\omega_4 = 2$ and $\omega_5 = 1$. Delays are affected by a penalty factor higher than one, which is linked to the priority. Since we are reporting delays multiplied by this factor, their values may appear high.

The models were trained on 100 randomly generated instances, with the fixed rail network described above, and tested on 100 unseen instances with the same specifications. The training phase involved running 10 000 episodes for each algorithm. An episode is stopped either when all trains are at their last resource or after a 2-hour plan is produced. In this context, we define an episode as the production of a 2-hour plan. In Table 1 we show some statistics, like minimum, average and maximum delay, its standard deviation and the number of deadlocks.In Table 2, we report the number of instances won, drawn, and lost by each approach (Centralized, Decentralized and Q-Learning). We have a win when the delay found is smaller, a draw when it is comparable, and a loss it is higher with respect to the values obtained by the other approach in comparison. By looking at the results, we see the Centralized approach reaches a delay that is less than Decentralized in 44 instances, being comparable in 47, and higher in only 9 of them. Comparing Centralized and Q-Learning, we have that in 93 instances the Centralized reaches a delay that is less than Q-Learning, and similarly can be said for the Decentralized approach. This highlights the Centralized has the best performances on this test set. Figure 7 reports the performance profiles for the three algorithms w.r.t. the value of the delay obtained on 100 instances of the same distribution of the training set. Performance profiles have been used as proposed in [12]. Given a set of solvers $\mathcal{I}$ and a set of problems $\mathcal{P}$, the performance profile takes as input a ratio between the performance (i.e. the value of the weighted delay) of a solver $i \in \mathcal{I}$ on problem $p \in \mathcal{P}$ and the best performance obtained by any solver in $\mathcal{I}$ on the same problem. Consider the cumulative function $\rho_s(\tau) = |\{p \in \mathcal{P} : \ r_{p,i} \leq \tau\}|/|\mathcal{P}|$ where $t_{p,i}$ is the delay and $r_{p,i} = t_{p,i}/\min\{t_{p,i'} : i' \in \mathcal{I}\}$. The performance profile is the plot of the functions $\rho_s(\tau)$ for $s \in S$. Informally, the higher the curve the better the component. Here, the components are algorithms and their performance is determined by comparing the delays obtained on each instance. The graph shows the supremacy of deep architectures with respect to the simple linear one, which is reasonable due to the known complexity of the problem. In general, the Centralized model seems to perform better than the Decentralized one.

Table 1 summarizes some basic performance statistics. As one can see,

Table 2: Wins, draws and defeats on 100 network test instances.

Win/Draw/Loss	Centralized	Decentralized	Q-Learning
Centralized	-	44-47-9	93-0-7
Decentralized	9-47-44	-	96-0-4
Q-Learning	7-0-93	4-0-96	-

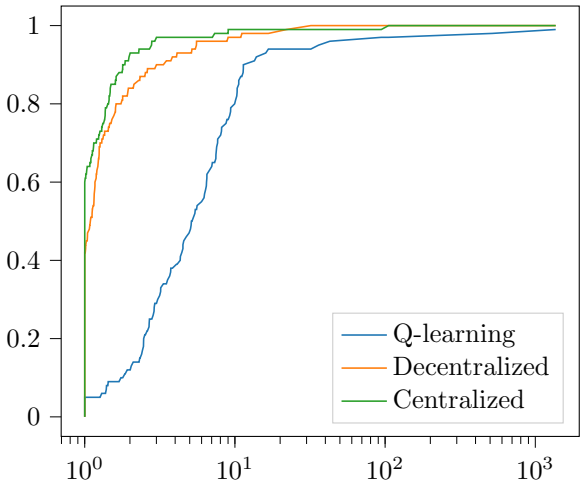

Figure 7: Performance profiles for the three algorithms compared together based on the value of the delay obtained on 100 instances of the same distribution of the training set.

the average delay of the `DeepRL` approaches is considerably smaller, so `DeepRL` captures inner non-linearities more efficiently and appears more effective than linear Q-learning.

5.2. Second experiment

As a second experiment, we test the ability of the three algorithms to generalize to longer trains with different frequencies. Longer trains in a network increase the probability to end up in a deadlock. In fact, long trains can occupy at the same time more than one resource and one or more switch. Even very simple configurations (e.g. one station with two stopping points and two crossing trains occupying both stopping points and switches) can lead to deadlocks. In other words, taking into account the length of trains introduces a whole new level of complexity, which is especially for freight-based traffic, where trains often tend to be very long. The time window considered is two hours.

In particular, for the test set the new parameters adopted are:

- Number of trains N; chosen randomly between 4 and 12 with the following probability distribution: $P(N = 4) = 0.05$, $P(N = 5) = 0.15$, $P(N = 6) = 0.15$, $P(N = 7) = 0.15$, $P(N = 8) = 0.15$, $P(N = 9) = 0.1$, $P(N = 10) = 0.1$, $P(N = 11) = 0.1$,$P(N = 12) = 0.05$

- Length; chosen uniformly in the set {4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000}, to be intended in feet

The focus is posed on these two parameters, since it has been observed empirically that they seem to be major factors in influencing the complexity of a `TD` problem.

In this case, the `DeepRL` Decentralized model is trained for 20 000 episodes, whereas the linear Q-learning for 50 000; we use the same learning of the previous experiment for the Centralized model.

The Q-learning and Decentralized `DeepRL` were trained on 200 instances from the distribution described in the first experiment (except for length and number of trains) and tested on 200 instances with the specifics specified above. Table 3 summarizes some basic performance statistics for this experiment. Both Decentralized approach and Centralized outperforms Q-Learning, moreover Centralized finds almost half deadlock than Decentralized and as in the previous experiment the average delay in Centralized is less than Decentralized. Linear Q-Learning appears to find less deadlocks than Decentralized approach. However, this is arguably induced by the fact that, in many test instances, the Q-learning based algorithm halts trains before they can reach a potential deadlock. This leads to huge delays, but falls short of explicitly creating a deadlock. In other words these results are somewhat biased, since in real-life a plan where trains are halted continuously (like those produced here by the linear Q-Learning approach) would be deemed equally unacceptable.
A last observation on the number of deadlocks, is that all the algorithms compared in Table 3 act as powered greedy heuristics. For this reason, their greedy nature may lead to unwanted solutions, which are more evident when the model cannot see the entire network, like in the Decentralized approach. We must not forget that the nature of the test instances is meant to consider also overcrowded situations, where a feasible solution may be hard to reach for a heuristic method. To the best of our experience, intensifying the training for longer periods or increasing the number of weights in the models may alleviate this circumstance.

Regarding the comparison among the different approaches, also in this case we appreciate the fact that both Centralized and Decentralized approaches outperforms the Q-learning. In Table 4, we can see that in 107 test instances on 200 the Centralized approach returns a delay that is less than the Decentralized one while in 182 test instances returns a delay that is less than the Q-Learning. Figure 8 shows the performance profiles in this case.

5.3. Third experiment: different sizes of network

In the last experiments, we test the generalization capability when increasing the number of resources in our network. We define three different railways: $R1$, $R2$, $R3$, with the same time window of two hours. Some characteristics of the railways are shown in Table 5. Railway $R1$ has the following parameters in terms of train traffic characteristics:

- Number of trains N; chosen randomly between 4 and 12 with the following probability distribution: $P(N = 6) = 0.05$, $P(N = 7) = 0.1$, $P(N =$

Table 3: Basic Statistics on 200 Test Instances from the Same Distribution of the Training.

Delay	Linear Q-learning	Decentralized	Centralized
minimum	0	0	0
average	50573.49	13966.1	13012.775
maximum	256883.1	86309.01	54979
std dev	52318.19	14092.37	12045.295
# deadlocks	18	26	14

Table 4: Wins, draws and defeats on 200 Test Instances from the Same Distribution of the Training.

Win/Draw/Loss	**Centralized**	**Decentralized**	**Q-Learning**
Centralized	-	107-70-23	182-3-15
Decentralized	23-70-107	-	189-3-8
Q-Learning	15-3-182	8-3-189	-

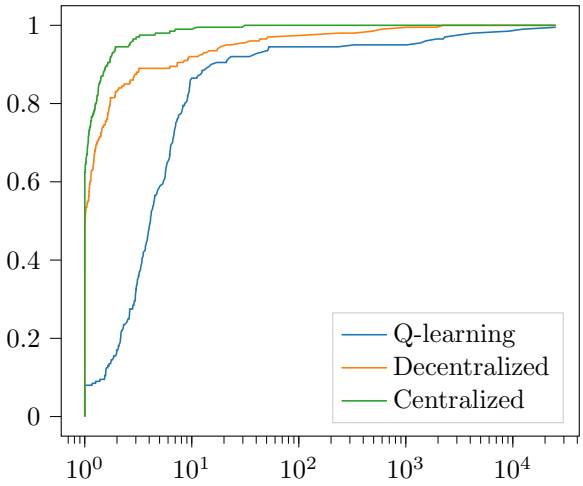

Figure 8: Performance profiles for the three algorithms compared together based on the value of the delay obtained on 100 with higher number of trains (10 to 15) than the training set.

Table 5: Railway network characteristics

Railway	Total resources	Stopping points	Single track resources	Double tracks
$R1$	155	79	52	12
$R2$	180	96	60	12
$R3$	196	106	60	15

8) = 0.1, $P(N = 9) = 0.15$, $P(N = 10) = 0.15$, $P(N = 11) = 0.1$, $P(N = 12) = 0.1$, $P(N = 13) = 0.1$

- Length; chosen uniformly in the set {4000, 4500, 5000, 5500, 6000, 6500,

Table 6: Basic Statistics on a new railway with 155 resources

Delay	Linear Q-learning	Decentralized	Centralized
minimum	1572	0	0
average	54611.55	16597.912	14074.175
maximum	153198.146	51952.006	47270.46
std dev	43666.76	14144.157	12861.334
# deadlocks	5	5	5

Table 7: Basic Statistics on a new railway with 175 resources

Delay	Linear Q-learning	Decentralized	Centralized
minimum	5412	1074	392
average	68347.693	27082.873	17157.326
maximum	246231.062	112981.007	52735.47
std dev	48582.189	24898.419	14324.176
# deadlocks	4	11	14

7000, 7500, 8000}

Railways $R2$ and $R3$ have the following parameters in terms of train traffic characteristics:

- Number of trains N; chosen randomly between 4 and 12 with the following probability distribution: $P(N = 7) = 0.05$, $P(N = 8) = 0.1$, $P(N = 9) = 0.15$, $P(N = 10) = 0.15$, $P(N = 11) = 0.15$, $P(N = 12) = 0.15$, $P(N = 13) = 0.1$,$P(N = 14) = 0.1$, $P(N = 15) = 0.05$
- Length; chosen uniformly in the set {4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000}

We generated 50 tests for each railway configuration. We did not train our networks on these new infrastructures, and we rather use the one adopted in the first experiment. In tables 6-7-8, some statistics for the three approaches. As shown in the tables, the two Deep-Learning approaches outperform linear Q-Learning, while the Centralized one again obtains better results in terms of weighted delay with respect to the Decentralized algorithm. In Tables 9-10-11 we can see how Centralized approach in about one third of the instances reach a delay that is less than the Decentralized, and it is comparable for one fourth of them.

6. Mixed integer programming comparison

Finally, we decided to compare our results for the Centralized approach with those obtained using a mixed integer program (MIP). The formulation is taken

Table 8: Basic Statistics on a new railway with 196 resources

Delay	Linear Q-learning	Decentralized	Centralized
minimum	4846	0	0
average	61033.829	21250.22	16844.557
maximum	159952.036	61314	56500
std dev	39812.026	15701.39	12972.443
# deadlocks	7	9	11

Table 9: Wins, draws and defeats on a new railway with 155 resources

Win/Draw/Loss	**Centralized**	**Decentralized**	**Q-Learning**
Centralized	-	32-11-7	47-0-3
Decentralized	7-11-32	-	47-0-3
Q-Learning	3-0-47	3-0-47	-

from [27] and it models both the routing and scheduling of trains. However, several of the features that were covered by our Reinforcement Learning model were not easy to extend in a mathematical formulation, therefore we had to neglect some of them. For example, a crucial aspect such as modelling the train length, and therefore its actual occupation of resources. The MIP model would have suffered computationally with this length adjustment, and it would have not been a fair comparison. For this reason, solutions of the MIP model with a lower value than RL solutions may potentially be infeasible. We report a brief description of the model adopted in [27] in the Appendix.

We ran the tests on the same machine of the previous experiments, and we report them in Table 12. To recap, the five different test sets where characterized by:

- 100 test instances on 137 resources (100 T 137 R)
- 200 test instances on 137 resources (200 T 137 R)
- 50 test instances on 155 resources (50 T 155 R)
- 50 test instances on 180 resources (50 T 180 R)
- 50 test instances on 196 resources (50 T 196 R)

We ran each test instance for 10 minutes. For each of the five test sets we consider:

Table 10: Wins, draws and defeats on new railway with 180 resources

Win/Draw/Loss	**Centralized**	**Decentralized**	**Q-Learning**
Centralized	-	29-16-5	39-0-11
Decentralized	5-16-29	-	41-0-9
Q-Learning	11-0-39	9-0-41	-

Table 11: Wins, draws and defeats on a new railway with 196 resources

Win/Draw/Loss	**Centralized**	**Decentralized**	**Q-Learning**
Centralized	-	30-14-6	44-0-6
Decentralized	6-14-30	-	46-0-4
Q-Learning	6-0-44	4-0-46	-

Table 12: Wins, draws and defeats on a new railway with 196 resources

Test	Solved (MIP)	Avg delay (MIP)	Avg delay 2 (MIP)	Avg delay (RL)	Both solved
100 T 137 R	98	8779	8716	9387	93
200 T 137 R	187	11712	11503	12893	173
50 T 150 R	37	23550	22960	14666	28
50 T 185 R	41	30339	27494	16936	28
50 T 197 R	43	32926	32504	16379	33

- the number of instances resolved in 10 minutes, where an instance is considered resolved if it finds an incumbent in the given time limit (Solved MIP).
- the number of instances solved both by MIP approach and RL, meaning the algorithm does not end with a deadlock (Both solved).
- the average delay of the sets according to MIP approach on the instances in the set *Solved MIP* (Avg delay (MIP)).
- the average delay of the sets according to MIP approach on the instances in the set *Both solved* (Avg delay 2 (MIP)).
- the average delay of the sets according to RL approach on the instances in the set *Both solved* (Avg delay (RL)).

As we can see, increasing the number of resources and/or trains the number of variables also escalates and therefore the number of solved instances solved reasonably decreases. At the same time, the average delay found in the MIP approach becomes higher than the ones found by RL. In fact while the values are lower for the first two test groups, they are significantly higher in the remaining sets. Particularly, for test sets 100 T 137 R and 200 T 137 R that have the less number of resources and trains MIP the average delay obtained by MIP approach is a little less than the Centralized approach. However when the number of resources increases (from 137 to 155, 180 and 196) and even the maximum number of trains is higher (from 10 and 12 to 13 and 15) MIP for test instances 50 T 185 R finds an average delay that is 150% higher than the average delay found by RL approach while for test sets 50 T 185 R and 50 T 197R the average delay found is twice higher than average delay returned by RL.

7. Conclusions

This study compares the use of Deep Q-learning with linear Q-learning for tackling the train dispatching problem. Two Deep Q-learning approaches were proposed: Decentralized and Centralized. The former considers a train as an agent with a limited perception of the rail network. The latter observes the entire network and uses a GNN to estimate the rewards, allowing to change the size of railway network without training a new neural network every time. Computational results inspired on data provided by a U.S. class 1 railroad show that the deep approaches perform better than the linear case. The generalization to larger problems, both in terms of number of trains and in terms of railway size, shows room for improvement, both in terms of search strategy and complexity of the tested network and instances. When the instance is generated by the same distribution of the training set, the algorithms prove to deal efficiently with the problem providing solutions in a very short time. This aspect is crucial in train dispatching, as indeed in any online/real-time planning problem, and in our opinion is one of the reasons that makes this research direction interesting. While solution algorithms based on different paradigms (e.g. Optimization, CP) have proven to tackle the problem effectively in certain cases, scaling and computational burden issues are always behind the corner. As shown in Section 6, we compared our Centralized approach with a MILP formulation, showing the discrepancy between the two methods grows with the complexity of the problem.i In future research, we may compare ourselves or include our heuristics into more complex models and algorithms like [26] and [10]. Moreover, Deep Reinforcement Learning (as in general ML-based approaches) has the advantage of shifting the computational burden to the algorithm's learning stage. Unlike in enumerative algorithms, online response time then becomes basically negligible. In other words, under the assumption that implicitly mapping space-actions-rewards (which presents its own, clear scaling issues) can be done effectively, Deep RL could represent a breakthrough in this application. For this reason, and others listed in the Contribution paragraph in Section 1, we believe that it is worth investigating and further advancing this research direction.

References

[1] B Adenso-Dıaz, M Oliva González, and P González-Torre. On-line timetable re-scheduling in regional train services. *Transportation Research Part B: Methodological*, 33(6):387–398, 1999.

[2] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *arXiv preprint arXiv:1910.07421*, 2019.

[3] Dimitri P Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA, 2019.

[4] M Boccia, C Mannino, and I Vasiliev. Solving the dispatching problem on multi-track territories by mixed integer linear programming. In *Proc. RAS Competition/INFORMS Meet.*, pages 1–16, 2012.

[5] Maurizio Boccia, Carlo Mannino, and Igor Vasilyev. The dispatching problem on multitrack territories: Heuristic approaches based on mixed integer linear programming. *Networks*, 62(4):315–326, 2013.

[6] Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014.

[7] Xiaoqiang Cai, CJ Goh, and Alistair I Mees. Greedy heuristics for rapid scheduling of trains on a single track. *IIE transactions*, 30(5):481–493, 1998.

[8] Alberto Caprara, Matteo Fischetti, and Paolo Toth. Modeling and solving the train timetabling problem. *Operations research*, 50(5):851–861, 2002.

[9] Yichen Chen, Lihong Li, and Mengdi Wang. Scalable bilinear π learning using state and action features. *arXiv preprint arXiv:1804.10328*, 2018.

[10] Francesco Corman, Andrea D'Ariano, Alessio D Marra, Dario Pacciarelli, and Marcella Samà. Integrating train scheduling and delay management in real-time railway traffic control. *Transportation Research Part E: Logistics and Transportation Review*, 105:213–239, 2017.

[11] Francesco Corman and Lingyun Meng. A review of online dynamic models and algorithms for railway traffic management. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1274–1284, 2014.

[12] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[13] Iddo Drori, Anant Kharkar, William R Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda Dietrich, David P Williamson, and Madeleine Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time. *arXiv preprint arXiv:2006.03750*, 2020.

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

[15] Taha Ghasempour and Benjamin Heydecker. Adaptive railway traffic control using approximate dynamic programming. *Transportation Research Part C: Emerging Technologies*, 2019.

[16] Andrew Higgins, Erhan Kozan, and Luis Ferreira. Heuristic techniques for single line train scheduling. *Journal of heuristics*, 3(1):43–62, 1997.

[17] Harshad Khadilkar. A scalable reinforcement learning algorithm for scheduling railway lines. *IEEE Transactions on Intelligent Transportation Systems*, 20(2):727–736, 2018.

[18] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[19] Leonardo Lamorgese and Carlo Mannino. An exact decomposition approach for the real-time train dispatching problem. *Operations Research*, 63(1):48–64, 2015.

[20] Leonardo Lamorgese, Carlo Mannino, Dario Pacciarelli, and Johanna Törnquist Krasemann. Train dispatching. *Handbook of Optimization in the Railway Industry*, pages 265–283, 2018.

[21] Jinlin Liao, Guang Yang, Shiwen Zhang, Feng Zhang, and Cheng Gong. A deep reinforcement learning approach for the energy-aimed train timetable rescheduling problem under disturbances. *IEEE Transactions on Transportation Electrification*, 7(4):3096–3109, 2021.

[22] Sundaravalli Narayanaswami and Narayan Rangaraj. Scheduling and rescheduling of railway operations: A review and expository analysis. *Technology Operation Management*, 2(2):102–122, 2011.

[23] Sundaravalli Narayanaswami and Narayan Rangaraj. Modelling disruptions and resolving conflicts optimally in a railway schedule. *Computers & Industrial Engineering*, 64(1):469–481, 2013.

[24] Lingbin Ning, Yidong Li, Min Zhou, Haifeng Song, and Hairong Dong. A deep reinforcement learning approach to high-speed train timetable rescheduling under disturbances. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3469–3474. IEEE, 2019.

[25] Mitsuaki Obara, Takehiro Kashiyama, and Yoshihide Sekimoto. Deep reinforcement learning approach for train rescheduling utilizing graph theory. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4525–4533. IEEE, 2018.

[26] Paola Pellegrini, Grégory Marlière, Raffaele Pesenti, and Joaquin Rodriguez. Recife-milp: An effective milp-based heuristic for the real-time railway traffic management problem. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2609–2619, 2015.

[27] Marcella Samà, Francesco Corman, Dario Pacciarelli, et al. A variable neighbourhood search for fast train scheduling and routing during disturbed railway traffic situations. *Computers & Operations Research*, 78:480–499, 2017.

[28] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[29] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[30] Darja Šemrov, Rok Marsetič, Marijan Žura, Ljupčo Todorovski, and Aleksander Srdic. Reinforcement learning approach for train rescheduling on a single-track railway. *Transportation Research Part B: Methodological*, 86:250–267, 2016.

[31] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[32] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Lanctot, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[33] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[34] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] Johanna Törnquist. Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms. In *5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[37] Johanna Tornquist and Jan A Persson. Train traffic deviation handling using tabu search and simulated annealing. In *Proceedings of the 38th annual Hawaii international conference on system sciences*, pages 73a–73a. IEEE, 2005.

[38] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9(2):187–212, 2022.

[39] Rongsheng Wang, Min Zhou, Yidong Li, Qi Zhang, and Hairong Dong. A timetable rescheduling approach for railway based on monte carlo tree search. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3738–3743. IEEE, 2019.

[40] Xuekai Wang, Andrea D'Ariano, Shuai Su, and Tao Tang. Cooperative train control during the power supply shortage in metro system: A multi-agent reinforcement learning approach. *Transportation Research Part B: Methodological*, 170:244–278, 2023.

[41] Yin Wang, Yisheng Lv, Jianying Zhou, Zhiming Yuan, Qi Zhang, and Min Zhou. A policy-based reinforcement learning approach for high-speed railway timetable rescheduling. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2362–2367. IEEE, 2021.

[42] C Yan and L Yang. Mixed-integer programming based approaches for the movement planner problem: Model, heuristics and decomposition. *Proc. RAS Problem Solving Competition*, pages 1–14, 2012.

[43] Lin F Yang and Mengdi Wang. Sample-optimal parametric q-learning using linearly additive features. *arXiv preprint arXiv:1902.04779*, 2019.

[44] Cheng-shuo Ying, Andy HF Chow, and Kwai-Sang Chin. An actor-critic deep reinforcement learning approach for metro train scheduling with rolling stock circulation under stochastic demand. *Transportation Research Part B: Methodological*, 140:210–235, 2020.

[45] Cheng-Shuo Ying, Andy HF Chow, Yi-Hui Wang, and Kwai-Sang Chin. Adaptive metro service schedule and train composition with a proximal policy optimization approach based on deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[46] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Appendix

We represent the railway network as a graph $G = (N, F, A)$ where each node $n \in N$ is a resource. Moreover we have two types of arcs: fixed and disjunctive. For each train and each routing, fixed arcs connects two subsequent resources while disjunctive arcs represent logic alternatives where at most one of them can be activated. A disjunctive arc represents precedence constraints on resources that can be shared by two trains.

The first group of variables is represented by the routing variables y_{um}, that are assigned to one if routing m for train u is chosen, and are zero otherwise. The second group are disjunctive variables $x_{(krj,ump),(umi,krp)}$, that are driven

to one if train k on routing r enters in resource p before train u that chooses routing m, and zero otherwise. Finally, the last group of variables t_{krj} are associated with time, in particular the instant when train k enters in resource j following routing r.

The resulting mathematical formulation is as follows:

$$
\begin{cases}
\min \quad \displaystyle\sum_{n \in T} t_n p_n & \\
\text{s.t.} \quad t_{krj} - t_{krp} \geq w_{krp,krj} + M(1 - y_{kr}), & \forall\ (krp, krj) \in F \quad (i) \\
\qquad t_{ump} - t_{krj} \geq w^A_{krj,ump} + M(2 - y_{um} - y_{kr}) + M x_{(krj,ump),(umi,krp)}, & \\
\qquad\qquad \forall\ ((krj, ump), (umi, krp)) \in A \quad (ii) \\
\qquad t_{krp} - t_{umi} \geq w_{krj,krp} + M(2 - y_{um} - y_{kr}) + M(1 - x_{(krj,ump),(umi,krp)}), & \\
\qquad\qquad \forall\ ((krj, ump), (umi, krp)) \in A \quad (iii) \\
\qquad \displaystyle\sum_{a=1}^{R_b} y_{ab} = 1, & \forall\ b \in \{1, \ldots, Z\} \quad (iv) \\
\qquad x_{(krj,ump),(umi,krp)} \in \{0,1\}, y_{um} \in \{0,1\}, t_k \in \mathbb{R}^+ &
\end{cases}
$$

Constraints (i) ensure that if train k chooses route r for each resource, then the starting time of two subsequent resources (i and $i+1$) must be at least equal to the time to travel the i-th resource. Constraints (ii) -(iii) are precedence constraints for trains that share the same resources. Group (iv) ensures each train uses just one routing.