Optimizing Train Scheduling with Learned Model

Lounès Meddahi
ENS Rennes
Rennes, France
lounes.meddahi@ens-rennes.fr

***Abstract*—We explore how Learned Models and Monte Carlo Tree Search (MCTS) can be used in the Combinatorial Optimization problem of train rescheduling. More exactly, we propose adapting DeepMind's MuZero algorithm to offer a scalable and time-efficient MCTS-based solution for train rescheduling from a single-agent modeling perspective. While MuZero and MCTS were developed for game-playing algorithms, we show that it can be used for the problem of train rescheduling. Our method leverages on a permutation invariant representation function to build, based on the train trajectories, the root node of the MCTS. This resusts in a learned representation that captures the complex dynamics of a railway system and able to explore its large action space. We demonstrate our approach on a simulation of Magdeburg's railway network. Our method learns to reschedule train journeys, providing more efficient scheduling in terms of delays caused by trajectories than simple baselines**

***Index Terms*—reinforcement learning, automated traffic management system, train rescheduling, monte-carlo tree search**

I. Introduction

Finding and generating optimized train trajectories, known as the train scheduling problem, and its underlying vehicle rescheduling problem (VRSP) [1], is a complex and old problem for both industrial and politics [2]–[6], as well as for scientists [7]–[14]. The increasing complexity and need for long distance transportation infrastructure [2], [15], especially railway network for goods and people [3], necessitates the development of effective automated action policies [16], [17], as human intervention to resolve recurrent local conflicts, such as traffic congestion on railway lines, becomes increasingly challenging [18]. The complexity of this problem is further amplified by the growing need for realistic simulation systems and the computational demands of analyzing and computing the consequences of actions on these simulators. This challenge, in addition of being an integer programming problem known for being NP-hard [19]–[23], often needs to be solved in real time, encouraging the use of quick heuristics.

To address the challenge of optimizing scheduled trajectories and reducing the overall delay, a task known as the train rescheduling problem, various approaches have been proposed. They typically rely on operational research techniques, single-agent systems, or multi-agent systems, with the goal of minimizing the global delay caused by trains across the network. However, the rapid expansion and increased complexity of train networks, such as with the adoption of moving blocks [30] to allocate trains to specific sections of the railway, further complicates the task of finding feasible and optimal solutions. This constraint hinders the scalability of traditional techniques for large-scale railway systems.

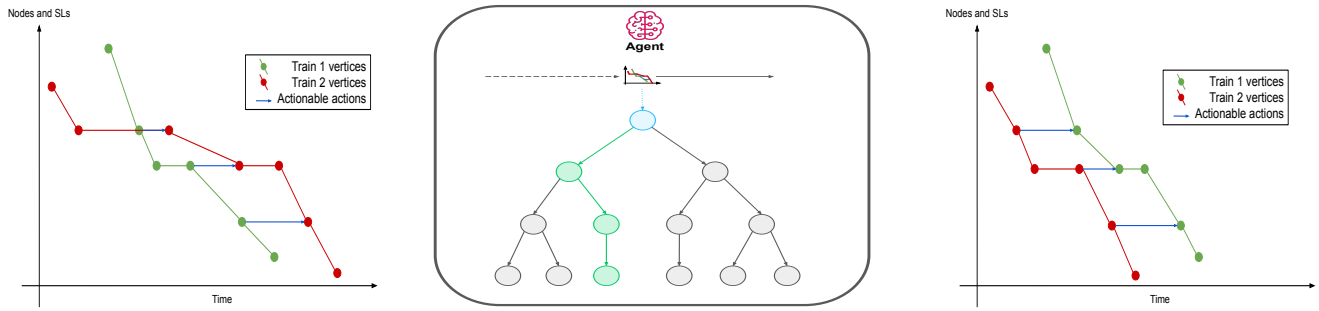
To address these limitations and ensure a feasible solution while focusing solely on events within the network that impact the delay, [32] applied Timed Event Graphs (TEGs) to represent these events within a Markov Decision Process (MDP) framework. Employing an MDP to model train scheduling using TEGs offers several advantages. First, it streamlines the problem's representation by offering a greater scope of the action space's size, compared to traditional methods that consider actions from all entities at each step (e.g. multi-agent systems). This aspect facilitates more efficient training and decision-making processes, especially for large-scale networks, as it allows to focus only on critical zones. Second, the MDP framework accommodates complex temporal dependencies between events. This enhanced representation enables for a reversible system and captures the dynamic nature of train trajectories movements and interactions, leading to a more informed decision-making process and improved performance under varying traffic conditions.

In this work, we introduce a new architecture based on Monte Carlo Tree Search (MCTS) and inspired by DeepMind's MuZero algorithm, tailored to the challenge of train rescheduling. Our approach, as shown in Figure1, involves iteratively learning and improving a permutation-invariant representation of the TEG's edge features, referred to as the latent state, through a recurrent process that utilizes both the latent state and the action to be performed on the system. We then use a prediction function and a dynamics function to build an MCTS that finds the most promising path, with the goal of optimizing train scheduling. To assist the MCTS's model in selecting the appropriate action at each iteration, the model predicts the value of the representation, its policy (e.g., the action to perform), and its immediate reward (e.g., the delay incurred by performing an action). The model is trained end-to-end with the sole objective of accurately predicting these three quantities, thereby enabling it to autonomously learn the edge representation it considers most effective for minimizing delay.

We evaluate the effectiveness of the learned policy using a simulator of Magdeburg's railway network. In this railway's simulator, we evaluate various performance metrics, including the average delay per train, the average delay per mission, the average handover delay, and the average relative gap.

To summarize, our contributions are:

(i) a new way to optimize train scheduling using learned model;



Non-optimal scheduling | Iterative rescheduling using our MCTS | Our optimized rescheduling result

Fig. 1: Given a Timed Event Graph (TEG), which represents trajectories of trains, we provide a more optimal scheduling strategy to reduce the overall delay using our MCTS-based solution.

(ii) a permutation invariant formulation of the representation function for the TEG's edges feature and;
(iii) a method that surpasses basic heuristics to optimize train scheduling.

II. Background

In this section, we briefly review concepts that are essential for understanding and describing our method in Section III, including Markov Decision Process notations and definitions (Section II-A), Reinforcement Learning (Section II-B), and Monte Carlo Tree Search (Section II-C), which form the foundation of our approach to solving the train rescheduling problem.

A. Markov Decision Process: Definitions and Notations

Our work focuses on training an agent to autonomously optimize train schedules within a railway network. The agent is situated in a simulated environment that represents a railway network, including the trajectories of the trains. The agent's interactions with this environment are iterative and discrete. At each time step $t \in \mathcal{T}$, the agent selects an action based on its current state. Subsequently, the environment responds with both a reward signal and an updated state.

We model this interactive process using a Markov Decision Process (MDP), defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$. Here, $\mathcal{S}$ represents the finite set of possible states of the environment, and $\mathcal{A}$ denotes the finite set of actions available to the agent. The reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, maps each state-action-next_state triplet to a real-valued reward. The transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ defines the probability of transitioning from one state to another, for a given action. Lastly, γ represents the discount factor of the cumulative reward, $R_t = \sum_{k \geq 0} \gamma^k r_{t+k}$.

B. Reinforcement Learning

In Reinforcement Learning (RL) [24], [25], [34], the goal is to assist an agent to learn a policy π to achieve a task, generally depicted by an MDP. The policy is a function mapping each state $s \in \mathcal{S}$ to a probability distribution over the action space $\mathcal{A}$. The agent aims to maximize its cumulative reward R over time by following and improving this policy π into an optimal policy π^*. At each step $t \in \mathcal{T}$, the agent has to choose between exploting its knowledge, by picking the action a such as $a = \max_a \pi(s_t, a)$, or exploring its environment to improve its policy, by picking the action $a = f(\pi(s_t, \cdot))$.

To do so, there are multiple ways such as using TD-Learning [26], Value Iteration [28], Policy Gradient Methods [29], and MCTS [35].

C. Monte Carlo Tree Search

Here we provide a brief overview of Monte Carlo Tree Search (MCTS). An in-depth survey can be found in [27]. MCTS stood out from other approaches by its application which revolutionized the game of GO [36], [37] and by high efficiency in challenging domains [38] (e.g. General Game Playing [39]). The main idea of an MCTS is to simulate hundreds or thousands of actions from the current state of the agent, using self-play, to find the most promising path towards the goal, generally pictured from the reward R. By simulating hundreds or thousands actions, the agent builds and expends a search tree, where the nodes are continuously updated to reflect how promising are they according to the goal. The evaluation of the nodes is done by computing a Q *value* to reflect the *Quality* of each node of the tree and by backpropagating this value to update the value of the nodes that led to this node. The MCTS works in four phases: selection, simulation, expansion and backpropagation. Those four phases are repeated such as the agent has built an MCTS to decide, from the root of the tree, what action is the most promising/valuable. Additionaly, this kind of tree search are best-first tree search procedure that means it can use heuristic such as neural networks to find promising solutions despite large and complex action space [40], [41].

III. Method

In this section, we begin by detailing the problem formulation we decide to solve, described in Section III-A. Then, we introduce DeepMind's MuZero algorithm, which lays the foundation of our approach.

A. Problem formulation

To solve the train rescheduling problem, we have decided to represent our problem using an MDP. The motivation for

employing an MDP in representing our problem lies in its ability to provide a model that reflects the true behavior of the system we want to study while minimizing the necessity for human-derived feature engineering. This approach not only facilitates an end-to-end problem-solving framework but also ensures a representation that closely mirrors real-world dynamics. Furthermore, such a representation inherently possesses the versatility to accommodate a diverse array of solution methodologies. This adaptability is crucial, as it allows for seamless application and comparison of different problem-solving approaches within the same foundational framework.

Our formulation for training the agent is grounded in the MDP framework as established in [32]. This framework is succinctly summarized in Table I.

TABLE I: MDP formulation of the train rescheduling problem

State	Timed Event Graph (TEG): • Vertices (Events): trains location, speed, delay, etc. • Edges (Processes): Processes that a train may undertake.
Action	Select an Edge, will trigger a re-order or re-route.
Observation	Global, edge, vertex features and action mask.
Reward	Difference of delay between S_t and S_{t+1}.

One of the primary challenges in creating and optimizing train scheduling using algorithms is ensuring the existence and feasibility of solutions. Traditional methods, particularly those using a multi-agent framework, often lack guarantees of producing feasible outcomes. [32] approach, however, is such a framework that it can begin with a straightforward heuristic like First-In, First-Out (FIFO), to offer a practical and viable starting point. Under this heuristic, the first train arriving at a junction is given precedence. Figure 2 illustrates how train trajectories are translated into a Timed Event Graph (TEG) using this methodology.

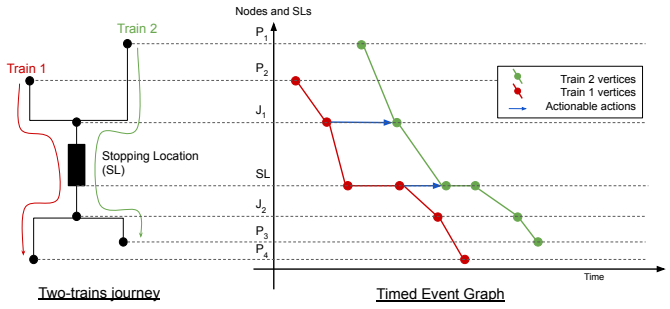


Fig. 2: Translation of two trains journey to a TEG.

In addition to guaranteeing a solution, if one exists, this representation offers several advantages. Firstly, it guarantees the environment to be reversible. Having a reversible environment brings guarantee that whenever the agent makes an action, this action consequences can be canceled by another action that brings the agent to its previous state. Secondly, as our goal is to reduce the overall delay of the network introduiced by the train trajectories, this representation highlits the two most important aspects of our problem:

- The temporal aspects of the trajectories during their journey
- The spatial locations where re-ordering and re-routing of trains are possible, and the nature of their interactions.

For all of the above motivations, we have decided to use [32] MDP modeling to modeled our trains scheduling problem.

B. MuZero for the Action Decision Process

As our problem modeling showcases directly the desired goal through the reward [33], we have decided to use RL to iteratively optimize train scheduling to achieve a rescheduling that results in a lower delay. The agent, at each iteration, has to decide what action to pick. To do so, we have decided to use an architecture based on MCTS.

However, any fixed representation of the environment is potentially sub-optimal, as it does not adapt to the underlying learning algorithm [31]. Furthermore, in the train rescheduling problem, the computation time to expand the search tree may increase with the number of trajectories, their length, and their interdependence. To deal with this, we would like to integrate to our solution a framework that also learns the dynamics of our problem to get a better understanding of it. To do so, we have decided to base our agent decision making algorithm on MuZero [42].

MuZero is an algorithm that uses a deep neural network μ_θ with parameters θ in order to build an MCTS. For each time step t, for each of $k = 1 \dots K$ steps, based on past observations, the model predicts three distinct quantities: the policy $\mathbf{p}_t^k \approx \pi(a_{t+k+1}|o_1, \dots, o_t, a_{t+1}, \dots, a_{t+k})$, the value function $v_t^k \approx \mathbb{E}[u_{t+k+1} + \gamma u_{t+k+2} + \dots |o_1, \dots, o_t, a_{t+1}, \dots, a_{t+k}]$, and the immediate reward $r_t^l \approx u_{t+k}$, where u is the true observed reward, π is the policy used to select real actions, and $gamma$ is the discount factor of the environment.

Internally and at each iteration t, MuZero works as a combination of three functions: a representation function, a dynamics function, and a prediction function. The representation function, $s^0 = h_\theta(o_1, \dots, o_t)$, initializes to root node of the MCTS by encoding the input historic of observations into a hidden state, named latent state. The dynamics function, $\left(r^k, s^k\right) = g_\theta(s^{k-1}, a^k)$, computes from the current latent state of the MCTS and the action to apply on it, the next latent state of the MCTS and the immediate reward associated to this transition. The goal of the dynamics function is to accurately replicate the behavior of the MDP dynamics associated with the problem. The prediction function, $\left(p^k, v^k\right) = f_\theta(s^k)$, computes the value and the policy associated to the internal state s^k in order to evaluate the state and assist to action decision. From the policy, the equation (1) is used to select

what action should be used to expand and explore the tree.

$$a_k = \arg\max_a \left[Q(s,a) + P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \left(c_1 + log\left(\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \right) \right) \right] \quad (1)$$

In order to guide MuZero's neural network μ_θ, the parameters θ are trained to approximate the policy, value, and reward, for every hypothetical step k, to corresponding target values observed after k actual time-steps have elapsed, using the loss (2) where the losses l^r, l^v, and l^p are loss functions for reward, value and policy respectively.

$$l_t(\theta) = \sum_{k=0}^{K} l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, p_t^k) + c||\theta||^2 \quad (2)$$

IV. Our approach

To provide a suitable and scalable approach to solve the problem, we propose a framework based on MuZero's algorithm and using [32] modeling.

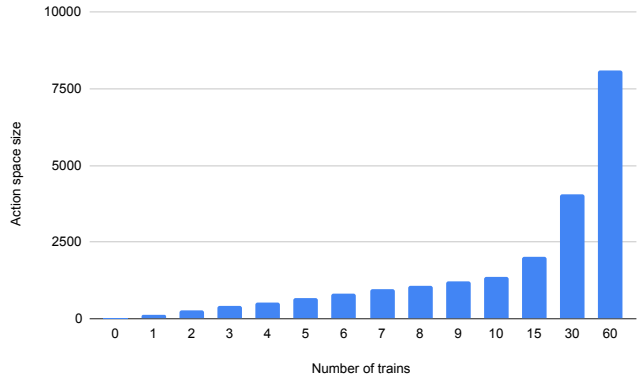


Fig. 3: This figure shows the growth of the action space size of Magdeburg's TEG for multiple number of trains.

A. Input of the framework

Based on a TEG representation of a set of trajectories, provided as detailed by [32], our goal is to use an RL framework that iteratively improve the trajectories scheduling in order to reduce the overall delay. As the input TEG is provided with all possible events of the network and all possible triggerable actions by masking the illegal ones, we have decided to keep only information that is directly related to our problem: How to optimize trains' trajectory in order to reduce the delay. To do so, we have decided to keep only the edge information of the TEG. Because the edges of the graph represent triggerable actions to re-route or to re-order trains journey, it means that having vertex features to know the trains' position can be avoided and lighten the input of the model as we have a masking of illegal actions. Because reducing the delay is our main target, and since the information is retained both in the vertex features and as the overall reward of the MDP, keeping only the overall reward to guide the model's training towards its task also lightens the model's input and makes learning easier.

Thus, from a given TEG representing a set of train trajectories, we finally end up keeping only the raw information and environment signals. This approach enables the model to learn its own representation of the environment's dynamics.

B. Proposed model

Because of the long time to compute and our large environment as demonstrated by Figure 3 , we based our model on MuZero's MCTS idea and keeping the same caracteristics of its functions:

- Representation function h:
  $h : \mathbb{R}^n \to \mathbb{R}^m$
  state $\mapsto$ latent state
- Prediction function f:
  $f : \mathbb{R}^m \to ([0,1]^{\mathcal{A}} \times \mathbb{R}^m)$
  latent state $\mapsto$ (Policy $\times$ Value)
- Dynamics function g:
  $g : \mathbb{R}^m \times \mathcal{A} \to \mathbb{R}^m$
  (latent state $\times$ action) $\mapsto$ latent state

However, we have adapted the model architecture of each function to use Multilayer Perceptrons (MLPs) [44]–[46]. This decision is based on the ease of use of MLPs and the promising results offered by MLPs architectures to learn any function to arbitrary accuracy [43]. Therefore, the representation function, the dynamics function, and the prediction function all utilize an MLP-based architecture.

However, to use the representation function, using the full TEG as an input would not have been a wise choice. We therefore use as input the input described in the Section IV-A. It would also allow the Neural Network to determine by itself the link between the journeys and the delay caused by these latters. That's why we have worked on an architecture able to learn a permutation-invariant representation of the edge features. Because there is no fixed and pre-determined position for the edge features within the graph, we have decided to apply the representation function to each edge features of the input TEG and then concatenate all outputs to get one single vector. We have, therefore, developed a restricted version of a Graph Neural Network (GNN) [47], [48].

Then, regarding the loss functions, we have decided not to use a Cross Entropy (CE) for the value and reward losses. This is because we decided against using a categorical representation for the value and reward outputs. Given that we don't have huge magnitude of the quantities to be predicted, we don't need a categorical representation to ensure that the scale of the gradient is independent of the magnitude of the value to be predicted [49], [50]. So, we have defined the losses using CE and Mean Square Error (MSE) as following:

- $l^p(\pi_{t+k}, p_t^k)$ = CE(π_{t+k}, p_t^k)
- $l^r(u_{t+k}, r_t^k)$ = MSE(u_{t+k}, r_t^k)

- $l^v(z_{t+k}, v_t^k)$ = MSE(z_{t+k}, v_t^k)

V. Results

In this section we present and analyze the results of our approach described in Section IV-B to the Magdeburg's city simulator, detailed by the Figure 4, same as [32] as a benchmark for train scheduling optimization.

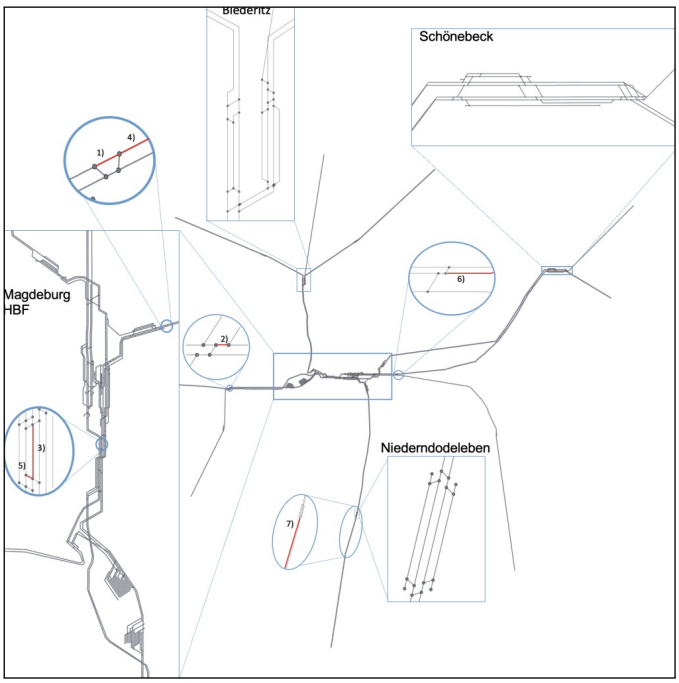


Fig. 4: *Magdeburg Infrastructure and blocked tracks in the evaluation suite:* The infrastructure consists of 29 stations, 354 stopping locations, 619 junctions, and 894 bidirectional tracks, with a total length of 343 kilometers. The enlarged rectangular sections correspond to stations. The blocked track for each scenario is highlighted in red and identified by the id of the scenario inside the enlarged rounded sections.

A. Details

To train our model, we used $K = 5$ hypothetical steps with a 800 simulations limit for the MCTS. The trainings proceeded batches of size 64, and the simulation ran for at least 25000 steps for a discount factor $\gamma = 0.99$. Because of the relatively low number of trains for this first evaluation of our approach, $n = 5$, we have made the supposition that we can use a historic size of 0, using only the current state as it is carries full information of the environment current state. The trajectories are initialized using *Raillation*, a greedy heuristic based on FIFO that orders trains based on their arrival time to the shared section. The representation function, which is composed of an MLP of depth 2 and size $[8, 4]$, takes one by one the edge features of each edge of the TEG and concatenate the results into one single vector. Regarding the prediction and dynamics functions, they use the same architecture that consists of an MLP of depth 4 of size $[512, 256, 128, 64]$, followed by a policy and value heads for the prediction function, and a head of size of $|E| \times 4$ for the dynamics function. *Rectified Linear Units* (ReLUs) [51] are used as *Rectified activation units* (rectifiers) between each MLP's layer.

The Figure 5 shows the performance throughout training on Magdeburg with 5 trains for various metrics.

B. Results analyzis

In order to evaluate the performances of our approach, we used the following metrics: Average delay per train, figure 5a, Average delay per mission, figure 5b, Average handover delay, figure 5c and the average relative gap, figure 5d.

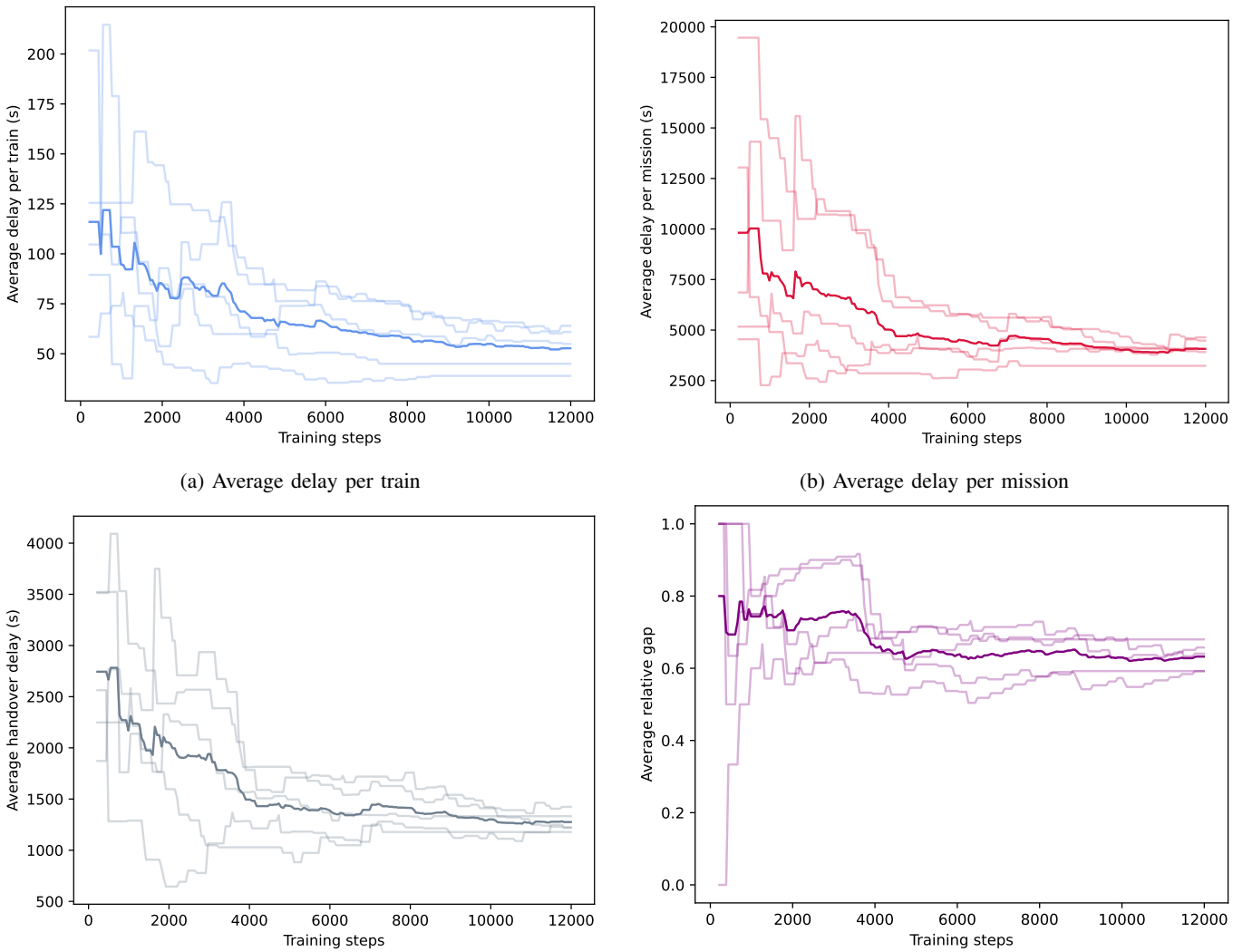
The mission delay, is computed by the sum of the delay at each mission the trains have during their trajectory. The mission delay of a train is then defined by $\sum_{\text{mission} \in \text{missions}} \frac{delay_{\text{mission}}}{|missions|}$. The average handover delay is computed by substracting the initial arrival time of the train by its real arrival time, so the handover delay of a train is simply $\text{train}_{\text{initial arrival time}} - \text{train}_{\text{real arrival time}}$. In can be seen as the natural delay of trains to accomplish its delay in the given railway network. The delay of a train is defined by the maximum delay it has for one of its mission plus its handover delay, $\max(\text{missions}_{\text{delay}}) + \text{handover delay}$. Finally, we have defined the relative gap mean. The relative gap mean, defined by the equation 3 is a metric that helps us to understand how well the model performs compared to delay caused by the network itself when considering the train trajectories one by one. To do se, we compute the minimum delay by summing the delay of the trajectory of each train if they were alone in the network. The initial delay corresponds to delay of the network at the initial step.

$$\min\left(\frac{\text{delay} - \text{minimum delay}}{\epsilon + (\text{initial delay} - \text{minimum delay})}, 1\right) \tag{3}$$

The Figure 5 shows the performance of our approach for the four describred metrics. As we can clearly see, our approach learns a policy that does what we want the agent to do: reduce the overall delay of the network, which was not a guarantee when using RL for a CO problem [52], [53]. With the Figures 5a, 5b, and 5c, the decreases are about 52.2%, 60%, and 52.6% respectively. By taking a look to the relative gap, which is the most indicative metric to know if the performances are good, has a drop of 21.75% going from a top of 0.8 to a minimum of 0.626.

Despite encouraging results, the last metric, the relative gap shown by The Figure 5d, suggests that the performances of our approach are not yet sufficient and satisfying. For only 5 trains, an impactfull result that oust current other best models for this same benchmark would have been a relative gap around 0.3.

Nevertheless, we can observe that the continuous and significant decrease of the delay metrics presented by The Figure 5a, 5b, and 5c result in one sharp drop of the relative gap around 4000 steps. Thus, despite being the most important metric, this unexpected behavior suggests that this metric should be more investigated before used to conclude regarding the efficiency of our approach.



(a) Average delay per train

(b) Average delay per mission

(c) Average handover delay

(d) Average relative gap

Fig. 5: **Evaluation throughout training on Magdeburg with 5 trains.** The x-axis shows the training steps until 12000 steps. The y-axis shows the delay in seconds for various metrics. We ran a total of 5 experiments and each experiment corresponds to 1 learner worker and 5 rollout workers. The rollout workers use the neural network trained by the learner to generate data, which are used by the learner to improve its neural network. The performance of the 5 experiments are showed by the translucent lines. The opaque one is the average of the 5 experiments.

VI. Related work

In this section, we give a brief overview of the few past and recent works tackling the train rescheduling problem using RL. Because OR approaches are not the subject of this work and have been explored for this problem for a long time, OR will not be addressed in this section. We focus on single-agent and multi-agent RL.

A. Multi-agent reinforcement learning

Multi-agent reinforcement learning (MARL) [54]–[56], is a subfield of RL where the focus is on each individual agent composing a system. In MARL, each agent is driven by its own reward and acts in its best interest. However, when applying MARL to the train rescheduling problem, it is not feasible to consider all trains simultaneously, as they must be considered separately as distinct agents. Thus, this approach makes it impossible to guarantee a feasible solution of trajectories at the initialization, even if one exists. Indeed, if we consider the example of *Flatland-RL* [57], a MARL benchmark for trains, it operates in a way that at each iteration, all trains must decide their next move. So, in addition to not allowing simple heuristics to ensure a solution, in cases with a large number of trains ($n > 5$), this representation may lead to situations where the agents never reach their objectives within the defined stopping conditions, especially in scenarios with long trajectories, high number of interactions between the trajectories, and numerous missions.

B. Single-agent reinforcement learning

On the other hand, the ability to consider all trains simultaneously, as proposed in [32], [60], [61], offers a solution for at least 5 trains and surpasses simple heuristics in performance for realistic simulators. In [61], the author propose an approach to this problem using instances that reflect actual lines of the Indian Railway. With their RL-based approach, they successfully scale to more than 400 trains while surpassing heuristics such as the Fixed Priority Travel Advance Heuristic [62] and the Critical First Travel Advance Heuristic [63]. In their paper, [60] employs *Q-learning* on the Netherlands railways to reschedule timetables offline, for up to 7 trains. In particular, they investigated the influence of the state representation on solution quality and solution convergence. In [32], where the simulation environment is the same to ours, they used Actor-Critic [59] and Proximal Policy Optimization [58] methods. Their experiments demonstrate successful top results, e.g. very low delay, with 20, 25, 30 and 35 trains.

VII. Conclusion

In this work, we introduced an MCTS and learned model-based approach to solve the Combinatorial Optimization problem of train rescheduling. We showed that this approach, despite yielding limited results compared to simpler RL algorithms such as PPO or AC, succeeds in learning policies that outperform the basic train scheduling heuristic, named *Raillation*.

However, further in-depth study is necessary. A more thorough understanding of our algorithm limits could be achieved by analyzing additional metrics, such as the number of actions used in building the MCTS, the time taken for its construction, and a comparison with AlphaZero [64]. Additionaly, more hyperparameter tuning such as increasing the size of the MLPs is needed.

Acknowledgment

The authors thank InstaDeep and Digitale Schiene Deutschland without whom this work and the Capacity & Traffic Management System (CTMS) project would not have been possible. We especially thank the following people for their help and support throughout the process of this work, in alphabetical order: Daniel Tapia Martinez, Gereon Vienken, Imen Djibouti, Khalil Gorsan Mestiri, Minh Tri Truong, Rihab Gorsane, and Vincent Coyette.

References

[1] J.Q. Li, P. B Mirchandani, and D. Borenstein. “The vehicle rescheduling problem: Model and algorithms”. In *Networks: An International Journal*, 50(3), pp.211-229, 2007.

[2] Giulio Mattioli. “Long-distance travel”. 2023.

[3] International Energy Agency. “The Future of Rail”. 2019.

[4] Arup Rail. “Future of Rail 2050”. 2019.

[5] International Energy Agency. “The Future of Rail, Opportunities for energy and the environment”. 2019.

[6] H. Huang, J. Xiong, and J. Zhang. “Windows of Opportunity in the CoPS’s Catch-Up Process: A Case Study of China’s High-Speed Train Industry”. 2021.

[7] H. Zhang, and S. Ni. “Train Scheduling Optimization for an Urban Rail Transit Line: A Simulated-Annealing Algorithm Using a Large Neighborhood Search Metaheuristic”. *Journal of Advanced Transportation*, vol. 2022, 9604362, 2022.

[8] V.F. Hurdle. “Minimum cost schedules for a public transportation route-I. Theory”. *Transportation Science*, 7(2), pp.109-137, 1973.

[9] P. Serafini, and W. Ukovich. “A mathematical model for periodic scheduling problems”. *SIAM Journal on Discrete Mathematics*, 2(4), pp.550-581, 1989.

[10] A. Higgins, E. Kozan, and L. Ferreira. “Optimal scheduling of trains on a single line track”. *Transportation Research Part B: Methodological*, 30(2), pp.147-161, 1996.

[11] M.R. Bussieck, T. Winter, and U.T. Zimmermann. “Discrete optimization in public rail transport”. *Mathematical Programming*, 70(1-3), pp.415-444, 1997.

[12] J.F. Cordeau, P. Toth, and D. Vigo. “A survey of optimization models for train routing and scheduling”. *Transportation Science*, 32(4), pp.380-404, 1998.

[13] A. Caprara, M. Fischetti, and P. Toth. “Modeling and solving the train timetabling problem”. *Operations Research*, 50(5), pp.851-861, 2002.

[14] V. Cacchiani and P. Toth. “Nominal and robust train timetabling problems”. *European Journal of Operational Research*, 219(3), pp.727-737, 2012.

[15] European Technology Assessment Group. “The Future of European long-distance transport, Scenario Report ”. European Parliament, 2008.

[16] Digitale Schiene Deutschland. “Digitale Schiene Deutschland develops an AI-based capacity and traffic management system using the ”Deep Reinforcement Learning” method”. 2020.

[17] Digitale Schiene Deutschland. “Artificial Intelligence as a game changer for Capacity and Traffic Management in the future railway system”. 2022.

[18] Digitale Schiene Deutschland. “Artificial Intelligence in the Capacity and Traffic Management System”. 2023.

[19] X. Cai, and C.J. Goh. “A fast heuristic for the train scheduling problem”. *Computers and Operations Research*, 21(5), pp.499-510, 1994.

[20] A.I. Mees. “Railway scheduling by network optimization”. *Mathematical and Computer Modelling*, 15, pp.33-43, 1991.

[21] C.J. Goh, and A.I. Mees. “Optimal control on a graph with application to train scheduling problems”. *Mathematical and Computer Modelling*, 15, pp.49-58 1991.

[22] R.G.J. Mills, and S.E. Perkins. “Nonlinear programming applied to the dynamic rescheduling of trains”. Internal report, Department of Mathematics, South Australia Institute of Technology, 1989.

[23] M.R. Garey and D.S. Johnson. “Computers and Intracrabiiity: A guide to the Theory of NP-Completeness”. Freeman, San Francisco, 1979.

[24] L.P. Kaelbing, M.L. Littman, and A.W. Moore “Reinforcement learning: a survey”. *Journal of AI research*, 4, pp.237-285, 1996.

[25] M. Wiering, and M. van Otterlo. “Reinforcement Learning”. Springer, 2012

[26] R.S. Sutton. “Learning to Predict by the Methods of Temporal Differences”. *Machine Learning*, 3(1), pp.9-44, 1988.

[27] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez-Liebana, S. Samothrakis, and S. Colton. “A Survey of Monte Carlo Tree Search Methods”. In *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), pp.1-43, 2012.

[28] M.L. Puterman. “Markov Decision Processes: Discrete Stochastic Dynamic Programming”. *John Wiley & Sons, Inc.*, New York, NY, USA, 1st edition, 1994.

[29] R.J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. *Machine Learning*, 8(3-4), pp.229-256, 1992.

[30] Digitale Schiene Deutschland. “Advanced Protection System”. 2023.

[31] Y. Tang, and R. Munos. “Towards a Better Understanding of Representation Dynamics under TD-learning”. In *International Conference on Machine Learning*, 2023.

[32] R. Gorsane, K. Gorsan Mestiri, D. Tapia Martinez, V. Coyette, M.B. Makhlouf, G. Vienken, M.T. Truong, A. Söhlke, J. Hartleb, A. Kerkeni, I. Sturm, and M. Küpper.“Reinforcement Learning based Train Rescheduling on Event Graphs”. In *IEEE International Conference on Intelligent Transportation Systems*, 2023.

[33] D. Silver, S. Singh, D. Precup, and R.S. Sutton. “Reward is enough”. *Artifcial Intelligence*, pp.103535, 2021.

[34] R.S. Sutton, and A.G. Barto. “Reinforcement Learning: An Introduction, second edition”. *The MIT press*, 2018.

[35] R. Coulom. “Efficient selectivity and backup operators in monte-carlo tree search”. In *International Conference on Computers and Games*, pp.72-83. Springer, 2006.

[36] R. Coulom. “ Computing Elo ratings of move patterns in the game of Go”. In *International Computer Games Association Journal*, 30, pp.198-208, 2007.

[37] S. Gelly, and D. Silver. “Achieving master level play in 9 x 9 computer Go”. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pp.1537-1540, 2008.

[38] S. Gelly, and D. Silver. “Monte-Carlo tree search and rapid action value estimation in computer Go”. *Artificial Intelligence*, 175(11), pp.1856-1875, 2011.

[39] H. Finnsson, Y. Björnsson. “ Simulation-based approach to general game playing”. In *Association for the Advancement of Artificial Intelligence*, 8, pp.259-264, 2008.

[40] T. Vodopivec, S. Samothrakis, and B. Šter. “On Monte Carlo Tree Search and Reinforcement Learning”. In *Journal of Artificial Intelligence Research*, 60(2017), pp.881-936, 2017.

[41] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. “A Survey of Monte Carlo Tree Search Methods”. In *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 2012.

[42] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, D. Silver. “Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model”. In *Nature*, 588, pp.604-609, 2020.

[43] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. *Neural Networks* 2, pp.359-366, 1989.

[44] P.J. Werbos. “Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences”. PhD thesis, Harvard University, 1974.

[45] Y. LeCun. “Une procedure d’apprentissage pour réseau à seuil asymétrique”. In *Proceedings of Cognitiva*, 85, pp.599 - 604, 1985.

[46] D.E. Rumelhart, G.E. Hinton, R.J. Williams. “Learning internal representations by error propagation.”. In *Parallel Distributed Processing*, MIT Press, 1, pp.318-362, 1986.

[47] P. Veličković. “Everything is connected: Graph neural networks”. *In-Current Opinion in Structural Biology*, 79(102538), 2023.

[48] M.M. Bronstein, J. Bruna, T. Cohen, P. Veličković. “Geometric deep learning: grids, groups, graphs, geodesics, and gauges.”. arXiv preprint arXiv:2104.13478, 2021.

[49] M.G. Bellemare, W. Dabney, and R. Munos. “A Distributional Perspective on Reinforcement Learning”. In *International Conference on Machine Learning*, pp.449-458, PMLR, 2017.

[50] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. “Implicit quantile networks for distributional reinforcement learning”. In *International conference on machine learning*, pp.1096-1105, PMLR, 2018.

[51] A. Krizhevsky, I. Sutskever, and G.E. Hinton. In “Conference on Neural Information Processing Systems”, 2012.

[52] Y. Bengio, A. Lodi, and A. Prouvost. “Machine Learning for Combinatorial Optimization: a Methodological Tour d’Horizon”. *European Journal of Operational Research*, 290(2), pp.405-421, 2021.

[53] N. Grinsztajn. “Reinforcement Learning for Combinatorial Optimization: Leveraging Uncertainty, Structure and Priors”. PhD thesis, Université de Lille, 2023.

[54] M. Tan. “Multi-Agent Reinforcement Learning: Independent vs Cooperative Agents”. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, pp.330-337, 1993.

[55] K. Zhang, Z. Yang, and T. Başar. “Multi-agent reinforcement learning: A selective overview of theories and algorithms”. *Handbook of Reinforcement Learning and Control*, pp.321-384, 2021.

[56] S.V. Albrecht, F. Christianos, L. Schäfer. “Multi-Agent Reinforcement Learning: Foundations and Modern Approaches”. *MIT Press*, 2024.

[57] S. Mohanty, E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, G. Vienken, I. Sturm, G. Sartoretti, G. Spigler. “Flatland-RL : Multi-Agent Reinforcement Learning on Trains”. arXiv preprint arXiv:2012.05893, 2020.

[58] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. “Proximal Policy Optimization Algorithms”. arXiv preprint arXiv:1707.06347, 2017.

[59] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In *International Conference on Machine Learning*, 2018.

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

[61] H. Khadilkar “A Scalable Reinforcement Learning Algorithm for Scheduling Railway Lines”. In *IEEE Transactions on Intelligent Transportation Systems*, 20(2), pp.727-736, 2019.

[62] S.K. Sinha, S. Salsingikar, and S. SenGupta, “An iterative bi-level hierarchical approach for train scheduling”. *Journal of Rail Transport Planning & Management* 6, pp.183-199, 2016.

[63] H. Khadilkar. “Scheduling of vehicle movement in resource-constrained transportation networks using a capacity-aware heuristic”. In *Procedure American Control Conference*, pp.5617-5622, 2017.

[64] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. arXiv preprint arXiv:1712.01815., 2017.