Shunting Trains with Deep Reinforcement Learning

Evertjan Peer, Vlado Menkovski and Yingqian Zhang
Eindhoven University of Technology
Eindhoven, The Netherlands
e.peer@student.tue.nl, {v.menkovski, yqzhang}@tue.nl

Wan-Jui Lee
Maintenance Development, NS (Dutch Railways)
Utrecht, The Netherlands
wan-jui.lee@ns.nl

***Abstract*—The Train Unit Shunting Problem (TUSP) is a difficult sequential decision making problem faced by Dutch Railways (NS). Current heuristic solutions under study at NS fall short in accounting for uncertainty during plan execution and do not efficiently support replanning. Furthermore, the resulting plans lack consistency. We approach the TUSP by formulating it as a Markov Decision Process and develop an image-like state space representation that allows us to develop a Deep Reinforcement Learning (DRL) solution. The Deep Q-Network efficiently reduces the state space and develops an on-line strategy for the TUSP capable of dealing with uncertainty and delivering significantly more consistent solutions compared to approaches currently being developed by NS.**

I. Introduction

The Dutch Railways (NS) manage a fleet of around 2750 carriages with which they operate about 4800 daily train rides. During peak hours, most of the carriages are in use. However, at night and off peak hours, the NS needs to cope with a surplus of 'rolling stock' (trains). Therefore, they are stored in a *shunting yard*. On these shunting yards, activities such as conducting repairs and cleaning of the fleet also take place.

Planning movements and tasks on train service sites is a major challenge to the NS. Among others, incoming train units need to be matched with outgoing train services according to the timetable, maintenance and cleaning activities need to be scheduled and trains need to be routed over and stalled on the tracks of the shunting yard. The complex physical layout of shunting yards makes the parking and routing of trains already a complex task: many tracks are dead-ended, overtaking is not possible with trains, track lengths (and train lengths) vary and maintenance and cleaning tasks are typically only possible on dedicated tracks on the yard. An example of a shunting yard is shown in Fig. 1.

At the time of writing, human planners manually generate a step-by-step plan, following a set of service site specific heuristic rules and aiming for a flow of trains that is robust to disruptions during plan execution. However, both busier railroad occupation - which increases the problem complexity - and strategic reasons call for an automated planning approach to assist the planners.

In this paper we focus on the situation in which no service activities need to take place on the service sites. This means that we reduce the planning problem to determining a sequence of train movements, which makes sure each train that is requested by the network can be delivered on time. Note that parking the sequence of arriving trains on the set of tracks resembles the *Bin Packing Problem* which is known to be $\mathcal{NP}$-hard. Here, however, the trains need to be parked in such a way, that whenever a train of a certain type and composition is requested by the service schedule, there is an unobstructed path for that train to leave (no other train is blocking the way).

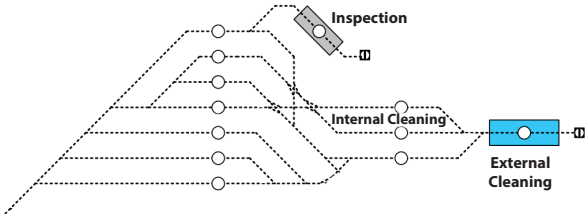


Fig. 1. Schematic illustration of a fictional shunting yard, with specific tracks for inspection and cleaning activities. Image adjusted from `www.sporenplan.nl`

This planning problem has been studied extensively in Operations Research (OR) literature, both including and excluding service activities and working with different shunting yard layouts, and is commonly known as the 'Train Unit Shunting Problem' (TUSP) [1] [2]. Exact solution approaches fall short when real-life problem cases are considered, since computation time increases drastically [3]. The local search based algorithm that is currently being developed at NS [2] is more successful in finding solutions to real-life problem cases. However, this approach assumes full information, thus ignoring uncertainties on arrivals and departures, and does not adhere to a step-by-step strategy. As a result, the movement plan seems ad-hoc to human planners, and solutions to the same problem instance can be very different when small perturbations occur. Because of these reasons, it is not easy for human planners to evaluate and adjust plans generated by this method.

Since in real life disturbances often occur in the planned sequences of trains, we aim to develop a method that creates consistent plans which are both predictable for human planners and robust to uncertainty in the arrival and departure sequence. When creating plans, the method should reason for each decision it takes which action is valid for most of the likely executions. Machine Learning, and more specifically the Reinforcement Learning (RL) framework, is a suitable technique to achieve this. For complex problems though, regular Reinforcement Learning falls short in learning strategies. Recent developments in the Deep Reinforcement Learning

(DRL) field, where RL and Deep Learning (DL) are combined, have shown that strategies can be derived for difficult problems where imperfect information is available [4][5][6]. As the TUSP is a sequential decision making process with delayed rewards, i.e., parking a train in front of another could lead to infeasible solutions later on, DRL is a suitable technique to address this problem. More specifically, since the TUSP is linked to a fixed physical infrastructure, we apply the DQN [5] which is able to take advantage of spatial relations when mapping images to values of actions.

In this paper, we model the TUSP as a Markov Decision Process for which we design a visual state representation for the TUSP together with an appropriate action space and reward function (Section IV). To incorporate uncertainty, we assume that at each time step t we only know the train compositions of the first m_a arrival and the first m_d departure events. For arrival and departure events further in the future, we only know the total number of carriages. At time t we do not know the train type (and composition) of those trains.

We train a DQN agent using real-world instances generated by NS, and then compare the consistency of the plans generated by the DQN agent to those generated by the local search approach [2], using an entropy metric to measure the uncertainty of the parking location of each train type when occurring at different positions in the arriving sequence (Sections V and VI). We find that at a small cost in terms of a lower number of solved instances, the DRL agent is able to find much more consistent strategies than the local search approach. This shows Deep Reinforcement Learning proves to be a promising technique to explore further in the context of sequential decision making problems in Operations Research.

II. TUSP FORMULATION

This research focuses on a limited version of the Train Unit Shunting Problem without service activities. A fictional shunting yard layout is used that is a simplified 'shuffleboard' structure, which also occurs in practice. In such an infrastructure the shunting yard consists of n dead-ended tracks with different lengths on which trains can be parked. In our experiments we will assume $n = 9$.

Central to the TUSP are trains. Trains can be composed of one or more train units of the same type, which are a set of carriages that form a self-propelling vehicle that can drive in both directions. Of the same train unit type, there exist multiple subtypes, where the subtype indicates how many carriages the train unit consists of. Fig. 2 shows the two subtypes VIRM4 and VIRM6 of the train type VIRM.

In this research we assume train compositions of a maximum of two train units and we assume there exist two train types (SLT and VIRM), which have the subtypes (SLT4, SLT6) and (VIRM4, VIRM6). Under these assumptions, there exist 10 unique train compositions in which trains can arrive at the shunting yard. For the departures, we assume that all trains consist of only one train unit. Implicitly, this means that if a combination of train units arrives, we directly decouple them after they are parked, since all departing trains will be separate

Fig. 2. Subtypes of train unit type VIRM: VIRM4 (top) and VIRM6 (bottom)

train units. In our setting, it is not possible to relocate a train temporarily from one track to another.

This restricted setting leaves us with the problem of finding 1) an assignment of all the arriving trains to parking tracks in such a way that no violation of track lengths occurs, and 2) an assignment of parked trains to departing services such that the correct type of train is assigned to the departing service and that all assigned trains have an unobstructed path to leave (no other trains are blocking the way out).

The following elements define *one* instance of the TUSP:

1) a timetable consisting of planned arrival of trains (both the composition and specific train units are specified),
2) a timetable consisting of required departures of trains (here only the composition is defined), and
3) a physical layout of a shunting yard.

The fact that for departures only the composition is defined implies that we are free to choose which exact train unit of the defined type we will assign to the departing service. So, if we have two VIRM4 train units on our shunting yard and one VIRM4 is requested, we are free to choose which one will leave for this service.

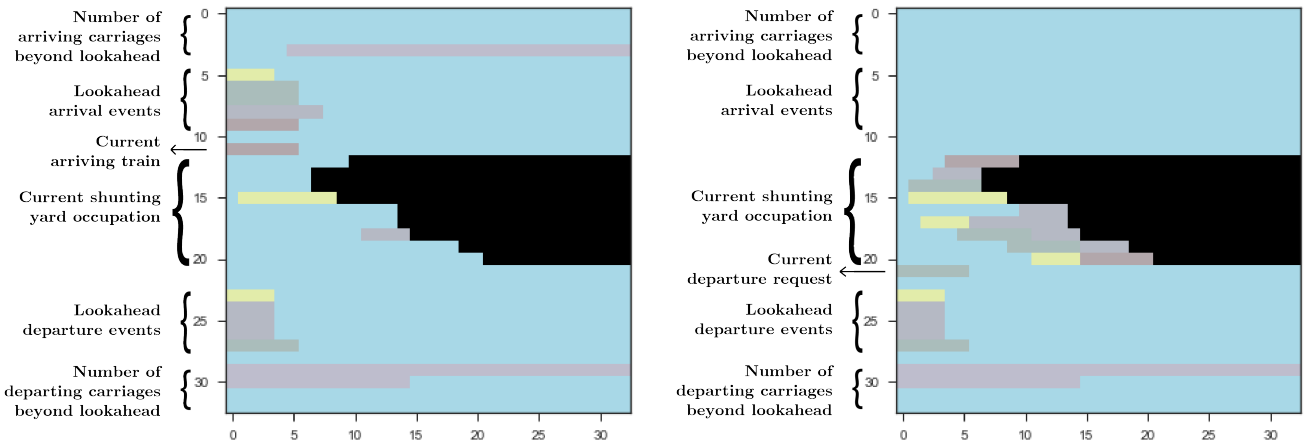
As mentioned in the introduction, we include uncertainty in our formulation by assuming that at each time step t we only know the train compositions of the first m_a arrival and the first m_d departure events. In our experiments we assume $m_a = m_d = 5$.

III. (DEEP) REINFORCEMENT LEARNING FRAMEWORK

Central to the **Reinforcement Learning** (RL) framework is an agent (actor) that is interacting with a certain environment. The agent observes the current state of the environment s and decides which action a to take. As a result, the environment moves to a new state s'. At this point, the environment provides a reward r for the action taken. The goal of the agent is to optimize its actions in order to achieve the highest cumulative reward over the relevant time horizon. Since the agent does not know anything about the environment, it can only learn from the rewards it observes during training. We refer to [7] for a more detailed introduction to RL.

Q-learning [8] is a *tabular* RL algorithm that assumes that the state-action space can be explored fully [9]. For each possible state and action pair, (s, a), Q-learning learns an estimate $\hat{Q}(s, a)$ of the optimal state-action value $Q^*(s, a)$. It has been shown that these estimates converge to the optimal values $Q^*(s, a)$ under the appropriate settings. However, as all state-action values need to be estimated separately, this is not practical in large state spaces.

Deep Reinforcement Learning (DRL) combines Deep Learning with RL, for instance by using deep neural networks



(a) Example of state with arriving train

(b) Example of state with departure request

Fig. 3. Visual state space design of arrival and departure states for a shunting yard with $n = 9$ tracks. The rows with black pixels denote tracks; trains can only be parked on the non-black pixels. Each carriage has a length of 1 pixel, and every train type has its own color encoding. At time t, the state encodes information about the current shunting yard occupation, the currently arriving or requested departing train, a lookahead of $m_a = 5$ arrival and $m_d = 5$ departure events and information on the total number of carriages that will arrive and depart beyond the $m_a = m_d = 5$ lookahead events.

as a function approximator for the Q-values of each state-action pair. This enables generalization from seen states to unseen states. However, using function approximation in the RL framework is known to be unstable. This is a result of the fact that RL by design collects correlated samples and the fact that targets are non-stationary, whereas deep neural networks rely on the assumptions of i.i.d. data distributions and stationary targets.

The **Deep Q-Network** (DQN) by [5] addresses these issues by using two techniques. First, they introduce *experience replay*, where the agent's experiences are stored in a data set and, when a Q-learning update iteration is executed, a sample of experiences is drawn uniformly at random from this data set. This removes the correlation of the data used to train the function approximator. Second, a separate neural network for the targets is used, which is only updated every fixed number of steps. By doing this, the targets are stationary for periods of time. In [5], the authors successfully parameterize an approximate value function $Q(s, a, \theta)$, using a deep convolutional network where θ denotes the weights of the neural network. Using convolutional neural networks as function approximator enables efficient learning in which spatial relations in the visual state representation can be utilized.

IV. DRL FOR TUSP

In order to apply the DQN techniques we formulate the TUSP in terms of a(n) (approximate) Markov Decision Process and design a visual representation of the state space.

1) Reinforcement Learning formulation: To fully define a Markov Decision Process we formulate $\mathcal{S}$: a finite state space, $\mathcal{A}$: the set of possible actions and $\mathcal{R}$: the reward function. $\mathcal{P}$, the transition probabilities, follows as a consequence of the problem instances we will generate for our experiments (as will be explained later). Thus, these probabilities are not explicitly defined here.

State space $\mathcal{S}$: In order to apply the DQN we have modeled a visual representation of the state space. We make a distinction between two types of states: arrival states (Fig. 3a) and departure states (Fig. 3b), in which respectively an arrival or a departure takes place. Each carriage of a train unit occupies 1 pixel in our state representation. Both arrival and departure states are defined by the following components:

Current shunting yard occupation: In each state, we know exactly which train units of which type are parked on which tracks. Together with the fixed track lengths, this is encoded in our state as a 9×33 image, which corresponds to the rows 12 to 20 in Fig. 3. The black areas symbolize the restriction to the length of the tracks. Trains can only be parked on the non-black areas of the 9 tracks.

Current arriving or requested departing train: Row 11 in Fig. 3a and row 21 in Fig. 3b show respectively the current arriving train and the train requested for departing. If the event is an arrival, row 11 will be occupied by a train and row 21 will be empty. For a requested departure, row 21 will contain a train, and row 11 will be empty, as visualized in Fig. 3.

5 lookahead arrival and departure events: We restrict the information in the states about the future to the coming $m_a = 5$ arrivals and $m_d = 5$ departures, which are included as rows 5 to 9 and 23 to 27 respectively in Fig. 3a and 3b. Note that this lookahead includes the current arrival or departure.

Number of carriages that will arrive and depart beyond the 5 lookahead events: Finally the state contains information about the number of carriages that will still arrive and depart beyond this lookahead window. No information is encoded about train types and order. For the arrivals, rows 0 to 3 are reserved for this in both state types in Fig. 3. For the requested departures, these are rows 29 to 32. The number of colored

(non-blue) pixels corresponds to the number of carriages that will still arrive/depart beyond the arrival/departure lookahead.

Action space $\mathcal{A}$: If the event is an arrival, the agent can choose an action a from $1, ..., n$ which corresponds to the track the arriving train should be parked on. If the event is a departure request, the agent again can choose a track. The train which is currently at the front of that track, will then be selected for departure.

Reward function $\mathcal{R}$: For this experiment we have constructed a simple value function (Equation 1).

$$r(s, a, s') = \begin{cases} 0.5 & \text{if correct parking} \\ 1 & \text{if correct departure} \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

A correct parking means no maximum track occupation is violated. A departure is only correct if the delivered train is of the correct (sub)type.

All these elements together formulate a finite Markov Decision Process. In this formulation we assume that the exact sequence of steps which led to the current state is not relevant for the algorithm to understand the current state.

2) Objectives and performance metrics: The DRL agent aims to maximize its expected reward over the complete time horizon. The way the problem and value function are formulated, implies that we are looking for one of the feasible solutions to the problem.

The performance of our method will be evaluated using two metrics. First, we calculate the percentage of instances in our test set for which a solution is found. Second, we introduce a measure of entropy, with which we quantify to what extend the algorithm follows a strategy when solving the problems.

For this second metric, we look at train subtypes, and in which step of the arrival sequence these types occur. We calculate per train type i and position j in the arrival sequence, the probability of being parked on each of the tracks k, for all successfully solved test cases combined. From the non-zero probabilities p_{ijk}, we calculate the entropy $E(i, j)$ using Equation 2. The entropy will be zero when it is sure that a certain type, when arriving in a certain position of the sequence, will be parked on a certain track. The highest entropy will be achieved when all $n = 9$ tracks are equally likely, which gives $E \approx 2.20$.

$$E(i, j) = -\sum_{k=1}^{n} p_{ijk} \ln(p_{ijk}) \quad (2)$$

We use the entropy measure as a proxy for plan consistency. A practical relevant measure of plan consistency is where trains of specific types are parked. Solving a plan using consistent strategies eases the planning of additional tasks that happen on a shunting yard, and allows for interpretability and ease of use for human planners.

3) Algorithm: Our aim is to bring the techniques developed by [5] to the Operations Research field by solving a real-life sequential decision making problem. Using the Reinforcement Learning framework we train an approximate value function $Q(s, a, \theta)$ based on experiences that the agent collects during training. We use a convolutional neural network which has an architecture similar to the one used in [5]. The input to the network is the 33×33 state representation as shown in Fig. 3. The first and second hidden layers are convolutional layers which convolve 32 and 64 filters of 4×4 and 2×2 respectively, both using a ReLU activation function. The third and final hidden layer is a fully connected layer with 256 units that also uses the ReLU activation function. The output layer is a fully connected linear layer with n outputs, one for each action. We use only two convolutional layers instead of three as used in [5] since our input dimensions are much smaller. In addition, we lowered the maximum experience memory pool-size in order to enable faster learning of the departure task, which only occurs halfway the problem for the first time. For full details on the DQN we refer to [5].

V. Experiment Setup

In order to evaluate our method, we use a problem instance generator developed by NS. This generator generates real-life scenarios of arriving and departure times and train compositions for the arrivals. All arriving activities occur before the first departing activity, which means that this number of train units is also the maximum number of train units that is present on the shunting yard at the same time. Table I shows the train units that are present in our instances and the approximate ratios of occurrence. Approximately half of the arrival events are trains composing of two train units, and half of one train unit. Departing trains always consist of one train unit.

To train the Reinforcement Learning agent, we generate $30,000$ problem instances with 14, 15, 16, and 17 train units ($7,500$ of each number of train units). At these numbers, the problem is reasonable complex: we need to park multiple trains on the same track in order to fit them all. However, we are sure that the total number of carriages is lower than the available space on the tracks. An example of an arrival sequence of 14 train units is: [SLT4, SLT4], [VIRM6], [SLT6], [SLT4], [SLT6, SLT4], [VIRM4, VIRM4], [VIRM4, VIRM6], [VIRM4, VIRM4], [VIRM4]. The brackets indicate trains, consisting of 1 or 2 train units.

Note that there are 10 possible arriving compositions (all units separately, and all combinations of two train units of the same type). There are over 3,5 million unique ways to generate a sequence of 9 train types out of those 10. For the instances with a higher number of trains this is even bigger, and we still do not take into account the departures in this computation. So, during training, the Reinforcement Learning agent only gets to see $< 1\%$ of the possible scenarios. To

TABLE I
Distribution of train subtypes occurring in arriving events

Traintype(subtype)	SLT(4)	SLT(6)	VIRM(4)	VIRM(6)
Length	4	6	4	6
Ratio	0.28	0.15	0.42	0.15

TABLE II
% SOLVED INSTANCES OF TEST SET FOR DIFFERENT INSTANCE SIZES.

	Problem instance size (in nr. of train units)			
Algorithm	**14**	**15**	**16**	**17**
Greedy	39.6%	37.5%	41.3%	37.0%
DRL	83.0% (2.2)	79.8% (2.8)	81.0% (3.2)	80.4 % (5.2)
[2]	91.1%	91.9%	92.8%	93.9%

test the agent's performance, we have generated another 750 problem instances for each of the four instance sizes, which gives a total of 3,000 test cases.

We will compare the scores on the two performance metrics of our algorithm to the scores of the local search algorithm with full information [2] and a simple greedy algorithm.

The greedy algorithm acts according to the following rules. If an arriving train consists of only 1 train unit type, the greedy algorithm will try to park it behind a train of the same type that already is present on the shunting yard. If that is not possible, and in case the train consists of multiple train unit types, the rule is to put the train on an available empty track. If there is no empty track available, the train will be put on a non-empty track that has enough available space for the train to fit. If no such track exists, the algorithm fails. For the departures, the greedy algorithm will check on which tracks there is a train of the requested type in front, and returns one of those. If no such track exists, the algorithm fails.

Whenever the greedy algorithm has multiple valid actions to choose from, it will pick the track with the highest number. This means the greedy algorithm will always park the first train on the last track, which corresponds to line 20 in Fig. 3.

VI. EXPERIMENT RESULTS

The algorithm is set up to train for 150,000 episodes. Every time a new episode begins, a random instance is drawn from the training data. To enable faster learning, the agent can try a maximum of three times to select a correct action from each state. An episode terminates when a problem has been solved, or when 3 times in a row, from the same state, an action has been chosen that resulted in a negative reward. This training procedure takes about 14 hours using the Intel®DevCloud. Note that our DQN implementation is not parallelized and thus only 1 CPU core is used.

a) Solved instances: We repeat our training procedure 4 times to create 4 agents. Their average score and standard deviation, together with the score of the local search approach and the greedy algorithm, are shown in Table II. The local search only fails when there exist no feasible solution to the problem: in that case it is not possible to solve the problem without a crossing to occur: a train that needs to overtake another. The DRL agents solve less problems, but still achieve a good score given the fact that each decision is made using imperfect knowledge about the future. The greedy algorithm only finds a solution to less than 40% of the problem instances in our test set. This underwrites that just using a very simple hand-crafted strategy is not good enough to solve the problem.

b) Plan consistency: Fig. 4 reveals the decisions made by the DRL agent for different train compositions, when these arrive first or second in the arrival sequence. The algorithm has made up its mind about preferred tracks for each train composition, including alternative choices depending on the current state. In Fig. 5 the entropy, as measure of certainty where a train will be positioned, is calculated for all train types and all steps in the arrival sequence. This is done for the DRL, local search and greedy solutions. In addition, the aggregated entropy measures for each train type (regardless of the position in the arriving sequence) are shown in Table III.

These results reveal that the DRL algorithm follows a strategy in which for 5 out of 10 possible arrival compositions, a practically fixed track assignment is used since the entropy is always close to zero. The other 5 arrival compositions with higher entropy scores are parked on a larger set of tracks, though in comparison with the results of the local search, the uncertainty of possible tracks is much smaller. Because of the random nature of the local search, it does not score much better than random allocation of arriving trains to tracks, which would result in an entropy score of $E \approx 2.20$. The greedy algorithm performs better than the local search, but worse than the DRL agent in terms of consistency on the instances that it solved. The entropy scores follow from the rules underlying the greedy algorithm. For instance, we find that the first train is always parked on the same track. Also, for the arriving train compositions that only contain 1 train unit type we find higher entropy scores, since these are, when possible, parked behind already present trains. It is obvious from these results that the DRL agent has derived different strategies than those that are implemented in this simple greedy algorithm.

Our experiments show that the DRL algorithm is consistent in parking arriving trains with the same composition on a limited set of tracks.

VII. CONCLUSION

In this work we have shown that benefits from the recent advancements in Deep Reinforcement Learning can be transferred to problems for which a visual representation of the state space is not obvious. We have shown that using a Deep Reinforcement Learning approach to the TUSP, a sequential decision making problem under uncertainty, leads to consistent solutions. The agent is able to find general strategies under uncertainty at a small cost in terms of a lower number of solved instances in the test set compared to the local search approach [2]. When used in practice, our method is preferred since it can be used in a step-by-step fashion, allows efficient (re)planning, and takes possible disturbances in the future into account. Deep Reinforcement Learning proves to be a promising method to explore further in the context of sequential decision making problems in Operations Research.

Future work directions regarding the application of DRL to the TUSP include scaling to problem instances with more trains and extending the problem formulation to include service activities and more complex shunting yard layouts. Also, more complex value functions can be introduced to prefer

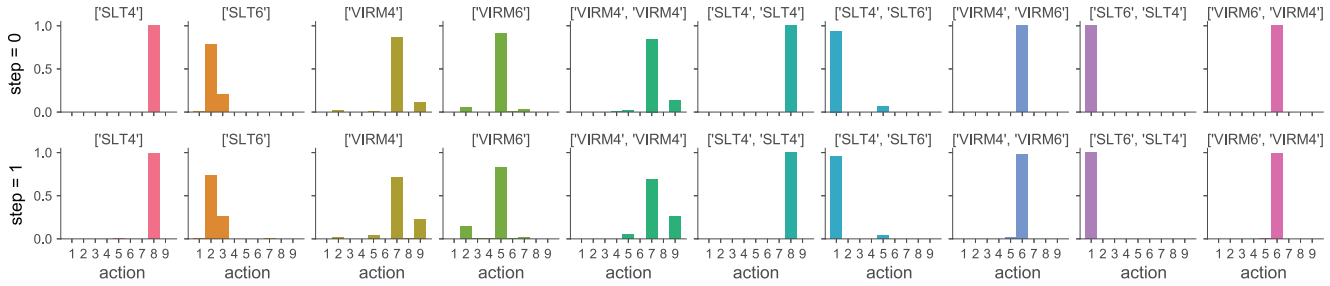


Fig. 4. Distribution of parking locations per train type for the first 2 steps by the DRL agent.

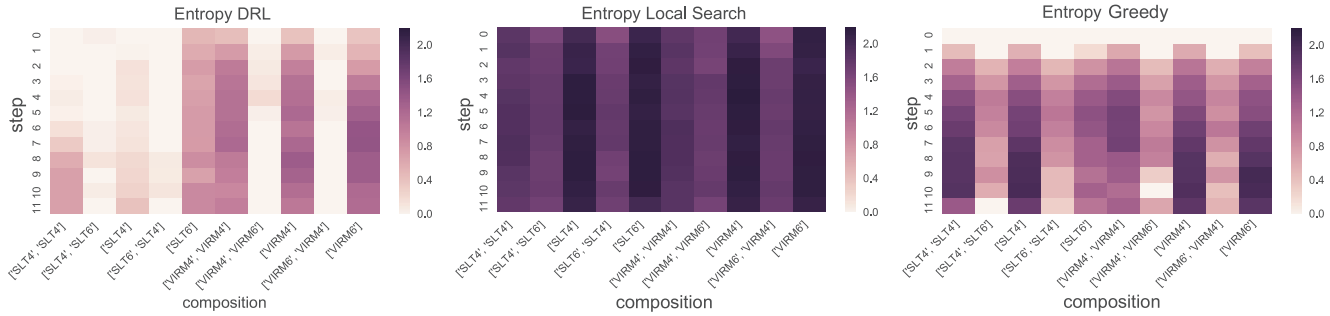


Fig. 5. Entropy of parking location decisions for the arriving train compositions.

TABLE III
AGGREGATED ENTROPY CALCULATION ON THE UNCERTAINTY IN PARKING LOCATION FOR EACH POSSIBLE ARRIVAL COMPOSITION.

	Train arrival composition									
Algorithm	SLT4,SLT4	SLT4,SLT6	SLT4	SLT6, SLT4	SLT6	VIRM4,VIRM4	VIRM4,VIRM6	VIRM4	VIRM6,VIRM4	VIRM6
DRL	**0.34**	**0.04**	**0.16**	**0.03**	**0.84**	**1.22**	**0.06**	**1.16**	**0.02**	**1.37**
[2]	1.93	1.79	2.18	1.78	2.19	1.92	1.78	2.19	1.77	2.18
Greedy	1.85	1.78	1.82	1.75	1.94	1.86	1.78	1.8	1.76	1.92

certain solutions over others. More advanced versions of the DQN could help to learn more efficiently when scaling, e.g. by introducing prioritized experience replay [10]. Also, other DRL methods such as the A3C (Asynchronous Advantage Actor Critic) [11] could be of help by enabling benefiting from parallel learners.

ACKNOWLEDGMENT

The authors thank Bob Huisman from NS for several helpful discussions. This work has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737459 (project Productive4.0). This Joint Undertaking receives support from the European Union Horizon 2020 research and innovation program and Germany, Austria, France, Czech Republic, Netherlands, Belgium, Spain, Greece, Sweden, Italy, Ireland, Poland, Hungary, Portugal, Denmark, Finland, Luxembourg, Norway, Turkey. The work is partially supported by the NWO funded project Real-time data-driven maintenance logistics (project number: 628.009.012).

REFERENCES

[1] L. G. Kroon, R. M. Lentink and A. Schrijver. *Shunting of passenger train units: an integrated approach.* Transportation Science, 42(4):436 449, 2006.

[2] R.van den Broek, *Train shunting and service scheduling: an integrated local search approach*, Master's Thesis, Utrecht University, 2016.

[3] F. Wolfhagen, *The Train Unit Shunting Problem with Reallocation*, Master's Thesis, Erasmus University Rotterdam, 2017.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis, *Mastering the game of Go without human knowledge.* Springer Nature, 550(7676): 354-359, 2017.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and Demis Hassabis, *Human-level control through deep reinforcement learning*, Nature, Springer Nature, 518(7540): 529-533, 2015.

[6] H. Mao, M. Alizadeh, I. Menache and S. Kandula *Resource Management with Deep Reinforcement Learning*, HotNets, 50-56, 2016.

[7] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st edition, MIT Press, 1998.

[8] C.J.C.H. Watkins and P. Dayan, *Technical Note: Q-Learning*, Machine Learning, 8(3): 279-292.

[9] M.L. Littman, *Reinforcement learning improves behaviour from evaluative feedback*, Nature, 521(7553):445-451.

[10] T. Schaul, J. Quan, I. Antonoglou and D. Silver, *Prioritized experience replay*, International Conference on Learning Representations, 2015.

[11] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*. Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016.