Semi-Markov Reinforcement Learning for Stochastic Resource Collection

Sebastian Schmoll and **Matthias Schubert**
LMU Munich
{schmoll, schubert}@dbs.ifi.lmu.de

Abstract

We show that the task of collecting stochastic, spatially distributed resources (Stochastic Resource Collection, SRC) may be considered as a Semi-Markov-Decision-Process. Our Deep-Q-Network (DQN) based approach uses a novel scalable and transferable artificial neural network architecture. The concrete use-case of the SRC is an officer (single agent) trying to maximize the amount of fined parking violations in his area. We evaluate our approach on a environment based on the real-world parking data of the city of Melbourne. In small, hence simple, settings with short distances between resources and few simultaneous violations, our approach is comparable to previous work. When the size of the network grows (and hence the amount of resources) our solution significantly outperforms preceding methods. Moreover, applying a trained agent to a non-overlapping new area outperforms existing approaches.

1 Introduction

A parking spot is often a rare resource and therefore, smart cities try to allocate them fair among all drivers. On that account, cities usually establish parking restrictions such as maximum parking duration. Unfortunately, people tend to violate restrictions. Hence, parking officers issue tickets for overstaying cars. In smart cities the assignment of parking space might be facilitated by the use of sensors which allow real-time monitoring of the current state of particular spots. The state of a parking spot can be *free*, *occupied*, *in violation* or already *fined*. Based on this information, it is not only possible to recognize violations but also to predict violations in the near future. Due to legal constraints, and special rules for residents, automated fining is not allowed. Thus, we aim to find a movement policy for an officer (single agent) which maximizes the number of issued tickets within working hours (finite horizon). The task is non-deterministic because overstaying cars might drive away before the officer arrives to record them. In previous work, this task is called Travelling Officer Problem (TOP) [Shao *et al.*, 2017].

Similar efforts exist in the transportation domain which can be outlined in the more general framework of Stochastic Resource Collection (SRC). The tasks vary with respect to observability, the dynamics of resources, and the stochasticity of rewards and/or travel times. Examples include the Taxi Dispatching Problem (TDP), and finding an available parking spot (Resource Routing). In the TDP, taxicabs are looking for passengers (resources) and may get information about current trip requests from a central entity. However, other cabs might serve the passenger first, or the passenger changes his/her mind. The major difference to the TOP is that after serving a passenger (or collecting a resource), the cab's position changes to the trip's destination. The Resource Routing task ends when a single resource is claimed. Though our proposed solution is generally applicable to any SRC, we focus on the TOP task in this paper. In previous work, the optimal policy for the TOP was approximated with solvers for a time-varying Travelling Salesman Problem (TSP). Here, we argue that SRC should not be considered as a TSP problem. Furthermore, the generalization of the TSP called Vehicle Routing Problem (VRP) has no existing suitable variation fitting to the examined problem setting of SRC tasks.The optimal SRC solution is non-deterministic because the transitions of the resources are typically unknown. Thus, the task should be modeled as a Markov Decision Process (MDP). Since the state space increases exponentially with the number of resources (or parking bays), finding an optimal policy using table-based solvers is infeasible. Therefore, we base our solution on Reinforcement Learning with function approximations. To handle non-uniform action duration (travel times), we propose to formulate SRC tasks as discrete-time Semi-Markov Decision Processes (SMDP). A challenge when learning an efficient policy for SRCs is to select a temporal abstraction for the action space. A instinctive choice would be to let the agent only decide among the next road segment at each crossing. However, based on this abstraction agents start with random-walks while training, which slowly explores the whole graph. Hence, we use higher temporal abstractions which directly consider traveling to any potentially useful location as action space. We present a novel neural network architecture developed for this abstraction which outperforms standard multi-layer perceptrons (MLP) by a large margin. Furthermore, our network architecture has a fixed size of parameters for any amount of resources. Hence, model sizes scale well with the amount of resources and trained agents are transferable to previously unseen regions. In our experimental setup, we compare our trained agent to the baselines proposed in [Shao *et al.*, 2017], and show that agents perform well when transferred to a new environment using real-world parking data from the city of Melbourne. Hence, our main contributions are (1) the first consideration of SRC as a SMDP at graph level and (2) solving it on a higher level temporal abstraction. (3) A novel, transferable neural network architecture adapted to this problem is introduced. (4) We compare our approach to existing baselines on a real-world setting.

2 Related Work

The general task of collecting resources has a broad range of real-world applications. [Shao *et al.*, 2017] studied the Travelling Officer Problem (TOP) where an officer moves within a street graph while maximizing the number of fined parking offenders. Since future development is uncertain, they propose a Greedy and an Ant Colony Optimization (ACO) approach. In their follow up work, both methods are approximated by imitation learning with an artificial neural network [Shao *et al.*, 2019]. Thus, the quality of the policies learned in [Shao *et al.*, 2019] approximate the solutions in [Shao *et al.*, 2017] but do not optimize the expected reward directly. If the future development of resources is deterministic, resource collection is a Vehicle Routing Problem (VRP) with time windows. A survey of VRP with time windows is given in [Solomon and Desrosiers, 1988]. In VRPs, the agent has to visit pickup and drop-off locations for cargo which are given in advance. In general, it is not acceptable for scheduled locations to be missed, therefore, solutions have to consider whether a route to all locations in the queue is still possible before choosing their next goal. One common subtask of VRP is the Travelling Salesman Problem (TSP). Recently, advances in solving TSPs have been achieved by using function approximations and reinforcement or supervised learning techniques [Kool *et al.*, 2018; Khalil *et al.*, 2017; Vinyals *et al.*, 2015]. In variations of the VRP with time windows, the agent does not know all customers at the beginning of the day, and over time more and more customers become known [Godfrey and Powell, 2002]. Although this task is dynamic as well, it differs from our setting in that the agent never knows the exact time windows. In VRP with stochastic customers [Gendreau *et al.*, 1996], it is unknown whether a customer is present but its demand is certain. In contrast, in our setting the availability of violations for the current point in time is known, but their state upon arrival is unrecognized. For Resource Routing (RR) [Schmoll and Schubert, 2018], an agent is moved to a single available resource in a highly dynamic environment. The classic application of RR is finding free parking spots or charging stations. This task can be defined as an MDP and solutions include dynamic programming [Schmoll and Schubert, 2018] and approximating the solution using replanning [Schmoll *et al.*, 2019] methods. The Taxi Dispatching Problem (TDP) is closest to our set-up. TDPs consider the dynamic nature of taxi customer appearing and disappearing if not served in time. Typically, TDPs aim at maximizing the number of served customers or minimizing the idle time of the taxi. Another property of TDPs is that serving a customer not only requires a certain time but also moves the agent to the drop off location. Within the literature, TDP is usually defined as multi-agent setting with either no observation [Kim *et al.*, 2019], or the common goal of all taxicabs is to distribute the fleet on a spatial grid or zone [Alshamsi *et al.*, 2009; Xu *et al.*, 2018; Lin *et al.*, 2018; Li *et al.*, 2019; Alabbasi *et al.*, 2019; Tang *et al.*, 2019]. The closest TDP approach to our work is described in [Tang *et al.*, 2019]. Albeit, this solution defines the TDP as a SMDP, it still work on a hexagon grid system and needs SMDPs for the options framework (meta-level actions). In comparison, we model the SMDP such that rewards appear at discrete time steps rather than assuming uniformly distributed rewards during the action execution. Furthermore, we formulate the optimal solution as a SMDP operating directly on the street network.

3 Background

A Markov Decision Process (MDP) $(S, A, R, \mathcal{T}, \gamma)$ consists of a set of states S, a set of actions A, a reward function R, a transition function $\mathcal{T}^a_{s,s'}$ defining the probability that s' is the follow state of s after executing action a. The discount factor γ is a value between zero and one defining the optimization horizon. Given an MDP, a deterministic policy π provides for each state $s \in S$ an applicable action $a \in A(s)$. A discrete-time, finite-horizon MDP has a pre-defined amount of equal-sized discrete time steps $t \in \{0, 1, \ldots, T\}$ available before an episode ends. It is important to distinguish between the MDP time step t and the time of the system ξ that may or may not be part of state $s \in S$. An agent being in state $s \in S$ at time t tries to find the action $a \in A$ that maximizes the future expected, discounted rewards, also known as state-value function V. The Bellman equation [Bellman *et al.*, 1957] defines a system of equations to compute the optimal values V^*:

$$V^*(s_t) = \max_{a \in A(s)} \sum_{s_{t+1} \in S} \left[\mathcal{T}^a_{s,s_{t+1}} \cdot (R(s_t, a, s_{t+1}) + \gamma V^*(s_{t+1})) \right] \quad (1)$$

Note that in this class of MDPs, the optimal policy π^* may not be stationary, as the horizon to optimize decreases with increasing t. In other words, the optimal action a for state s might be different for varying t. Often, it is beneficial to consider the state-value for a given action, i.e. state-action-value $Q(s_t, a)$. The Bellman equation for state-action values directly follows from (1) by omitting the maximum operator.

In many real-world applications, it is intractable or error-prone to define a transition function $\mathcal{T}$. Consequently, there are methods that learn the policy π, values (V / Q) or both from a simulation or the real-world without knowledge about the underlying model. These approaches fall into the class of reinforcement learning. Without loss of generality, we base our solution in this paper on Watkins' Q-Learning [Watkins, 1989], which iteratively runs trials in the simulation and updates the respective Q-value on every state observed with a learning rate α. See the formula below:

$$\begin{aligned} Q(s_t, a) \leftarrow{} & Q(s_t, a) + \\ & \alpha \left(r_{t+1} + \gamma \max_{a' \in A(s_{t+1})} Q(s_{t+1}, a') - Q(s_t, a) \right) \end{aligned} \quad (2)$$

Tabular Q-learning – i.e. every state-action-time triple (non-stationary policy) (or every $Q(s_t, a)$) has a dedicated trainable parameter – converges eventually to the optimal Q-values. However, in our application, the state space is too big for tabular representations. Furthermore, if every state has its trainable parameter, abstraction of observations is impossible. Consequently, the use of Function Approximations (FA) is often essential. Unfortunately, FA also leads to weaker convergence guarantees (e.g. Braid's counterexample [Baird, 1995] and the deadly triad [Sutton and Barto, 2018]). The authors of Deep-Q-Network (DQN) [Mnih *et al.*, 2013] introduced two techniques to soften the effects of FA mentioned earlier. First, they proposed a replay buffer where experiences are stored such that an artificial neural network can train randomly sampled mini-batches from the buffer. Among other advantages, using an experience buffer improves sample efficiency. Second, the DQN algorithm freezes the parameters for the target Q-function for a specific amount of training steps to reduce the moving target issue. Subsequently, several improvements, e.g. DoubleDQN [Van Hasselt *et al.*, 2016] and prioritized experience buffers [Schaul *et al.*, 2015], have been proposed to further improve the performance. Recently, there have been advances in policy gradient and actor-critic methods, e.g. [Mnih *et al.*, 2016; Lillicrap *et al.*, 2015; Schulman *et al.*, 2017]. Although, our architecture could be applied to policy gradient methods as well, for the sake of simplicity, we decided to focus on DQN approaches, as we have a discrete action set, and DQN is often more sample efficient.

4 Stochastic Resource Collection

In Stochastic Resource Collection (SRC), a set of resources $\mathcal{P}$, located on an edge $e \in E$ of street graph $G = (N, E, C)$, where N is the set of nodes and C are the travel costs, can be collected by an agent. Over time, a resource $\rho \in \mathcal{P}$ changes its property w of being collectible or not. The transitions of w are uncertain, and a precise model is not known. Whenever the agent traverses an edge $e \in E$, it gathers all collectible resources located at e. The goal of SRC is to maximize the collected resources in a given period by guiding the agent through the graph G.

The SRC defined above can be transferred into an MDP as follows. A state $s \in S$ is a concatenation of the agent's location and all resources states', including all information necessary for being Markov. The latter may depend on the actual real-world application. We define the action space as $A = E \cup \{a_c\}$, where a_c is a "continue" action, applicable to all states where the agent's location is an edge, not a node. Further, we say:

$$A(s) = \begin{cases} n.outgoing & \text{if } s \text{ on node } n \\ \{a_c\} & else \end{cases} \tag{3}$$

The real-world duration $\Delta\xi$ of a time-step d is the greatest common divisor of all edge travel times. Differently expressed, taking any action adds d to the environmental time ξ. The reward function R is incremented by 1 for every collected resource and otherwise 0. Due to the unknown dynamics of the resources' states, the transition function $\mathcal{T}$ is unknown.

5 Temporal Abstraction

In practice, one would not want to update parameters for states between two nodes, as the only valid action is the continue action a_c. Hence, the lowest-level temporal abstraction being practical is actions completely traversing edges. As those meta-level actions may vary in the number of required time steps, one needs to give special attention to the discount factor γ. This setting is called Semi-Markov-Decision-Process (SMDP). The SMDP consists of the same tuple as MDPs $(S, A, R, \mathcal{T}, \gamma)$. The only difference is the transition function $\mathcal{T}$. Instead of sampling the following state, given the current state and action, T additionally samples the time steps exhausted, i.e. $\mathcal{T}_{s,a}^{s',\tau} := P(s', \tau \mid s, a)$. Let's assume that executing action a took τ time-steps. Then, the update rule for Q-learning is [Barto and Mahadevan, 2003]

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left(\sum_{i=0}^{\tau-1} \gamma^i r_{t+i+1} + \gamma^\tau \max_{a' \in A(s_{t+\tau})} Q(s_{t+\tau}, a') - Q(s_t, a) \right). \tag{4}$$

Notwithstanding, there are other considerable abstractions, e.g. directly routing to a specific target. Although the policy can eventually learn to follow optimal paths, the learning process itself accelerates notably by outsourcing the well-understood shortest-path task to external solvers. The temporal abstraction used in our approach is outlined in Section 6.

5.1 Semi-Markov DoubleDQN

In our experiments, we use the DoubleDQN with prioritized experience replay buffer [Van Hasselt *et al.*, 2016; Schaul *et al.*, 2015]. Due to the variable action duration in our SMDP, the DQN algorithm needs slight adaption. Note that since our SMDP has discrete time events, there is no need to integrate the rewards as done in [Bradtke and Duff, 1995; Crites and Barto, 1998] even though the time step duration may be small. In particular, we adjust the update rule of the DoubleDQN to

$$Q(s_t, a, \theta) \leftarrow Q(s_t, a, \theta) + \alpha \left(\zeta + \gamma^\tau Q(s_{t+\tau}, a', \hat{\theta}) - Q(s_t, a, \theta) \right), \tag{5}$$

where θ are the FA parameters, $\hat{\theta}$ are the FA parameters for the frozen target network, a' is the optimal action w.r.t. θ, i.e. $a' = \arg\max_{a'' \in A(s_{t+\tau})} Q(s_{t+\tau}, a'', \theta)$, and ζ is the experienced time discounted reward provided by the environment, i.e.: $\zeta = \sum_{i=0}^{\tau-1} \gamma^i r_{t+i+1}$.

6 Environment

We examine our solution approach to SRC on the special case of the Travelling Officer Problem (TOP), as first defined in [Shao *et al.*, 2017]. In this problem setting, an officer tries to maximize the parking violation fined in a given area of Melbourne. Due to parking sensors, the officer knows in real-time whether a car is in violation or not. Note that this is a real scenario which is actively applied in Melbourne. We use the real-world and freely available dataset of on-street parking spots[1], containing the arrival and departure times as well as

[1]https://data.melbourne.vic.gov.au/browse?tags=parking

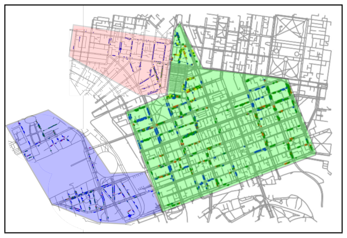
Figure 1: Illustration of the environment areas (blue: Docklands (613 resources), red: Queensberry (662 resources), green: Downtown (1954 resources)). Downtown is a collection of many tiny areas. Small colored dots are parking spots. Color denotes the state.

the respective restrictions of parking events of the Melbourne city in the year of 2017. Our environment uses this dataset as a replay to simulate the real-world as close as possible. To this end, we do an inner join of the table providing the parking events with another table containing the locations of each parking sensor. We extract a walking graph for the area from OpenStreetMap[2]. The preprocessing step thereafter assigns all parking spots to the closest edge in the graph. If an officer traverses an edge containing parking spots in state violation, he/she gets rewarded with $+1$ at the time the officer passes the resource. An officer's working day is from 7 a.m. to 7 p.m. and always starts at the same randomly chosen node. An illustration of the environment and the used areas are sketched in Figure 1.

The design of the actions' meta-level is an important decision for the environment. The most flexible level would be to let the agent decide the next edge. However, we decided against it and rather propose a more sample efficient action abstraction by outsourcing deterministic routing decisions to an external solver (Dijkstra). Thereby, the focus lies on deciding which of the edges containing resources, the officer should visit next. Hence, the action space used in our experiments is $A(s) := \{e \mid e \in E \wedge e \textit{ has resources}\}$, $\forall s \in S$.

For practical reasons, the greatest common divisor d of all edge travel times is not directly computed. Instead, we propose to use a scaled discount factor $\hat{\gamma}$ defined as $\hat{\gamma} = \gamma^{1/d}$. It is not necessary to compute neither d nor the actual discount factor γ. The benefit of using $\hat{\gamma}$ is, that the real valued times $\hat{t}$ can be used instead of t. Note that although the time is real valued, our SMDP resides on a discrete time SMDP. The discrete time t is merely scaled to $\hat{t}$. Furthermore, defining $\hat{\gamma}$ is more intuitive.

For the sake of simplicity, we assume the parking spot is passed at the end of the edge and tagging the car does not consume any time. A modification to a more realistic setting, including more complex reward functions, would not need any alteration to the main concepts of our solution approach. More precisely, in the simulation, ζ is defined as $\zeta = \sum_{(\rho,\Delta t)\in\mathcal{F}} \hat{\gamma}^{\Delta t}$. Here, $\mathcal{F}$ is the set of all parking spots fined $\rho \in \mathcal{P}$ plus the fining time point Δt relative to the start time of the currently executed action. Note that r is omitted as its value is $+1$.

[2]https://www.openstreetmap.org/

7 Function Approximation

The potential space of possible function approximations is vast.We present a sample efficient, transferable neural network architecture. This function approximation has to return a Q-value for each action, i.e. for each edge containing at least one resource. In the following, we present the architecture of our artificial neural network, whose parameter size does not increase with the amount of resources. Furthermore, parameters can be transferred to another, spatially not overlapping area. First, observation input is defined, followed by subsequentially explaining the next layers until the output layer is reached. An overview of the function approximation is sketched in Figure 2.

Observation. The observation, or the input of the function approximation, is a matrix X where each row X_i represents a resource from the environment. We take advantage of the fact that the routing part of the state transition is deterministic. The main stochasticity of the state transition are arriving and departing vehicles. Note that this approach is related to afterstates [Sutton and Barto, 2018]. We look up the precomputed distances to the resources and estimate the time of arrival. Then, we examine whether it is free, occupied, in violation or fined for the given time – assuming no arrival and departures. The first four columns in X_i represent a one-hot encoding of the aforementioned state. The next column is the walking time to the resource, such that the FA can assess the degree of stochasticity. Further, the current time of the day ξ and the time of arrival are included. Finally, we add a real-valued number between -1 and 2, indicating how long a car is still allowed to occupy the resource, and how long it is in violation, respectively. Having a score higher than zero shows a violation. Some information provided is redundant, but it appears that all information is useful for the agent. Not every resource behaves equally. For instance, there are parking spots for disabled, loading zones, etc. It is possible appending a resource-dependent embedding such that the agent can learn the resources' properties. However, no significant improvement could be found, while loosing the transferability to other areas.

Resource representation. The first step of our function approximation is to compute an h dimensional representation for each resource by using the observation X. Therefore, we use a standard MLP with one hidden layer and h output neurons f^h. The same parameter set θ_P is applied to all resources, i.e. $P_i := f^h(X_i, \theta_P)$ with ReLU as activation function.

Convolution. So far, the function approximation has one representation for each resource. However, our actions are all possible edges containing at least one resource. Therefore, the action space is potentially smaller than the resource dimension. One naive idea is to sum up the representations of all resources located at the same edge, and subsequently, predict the Q-value for each action. We decided against it, as one would ignore close by resources located at neighbouring edges. For this reason, we propose an approach similar to graph convolutions. First of, we compute a distance matrix D where each element d_{ij} represents the shortest-path distance

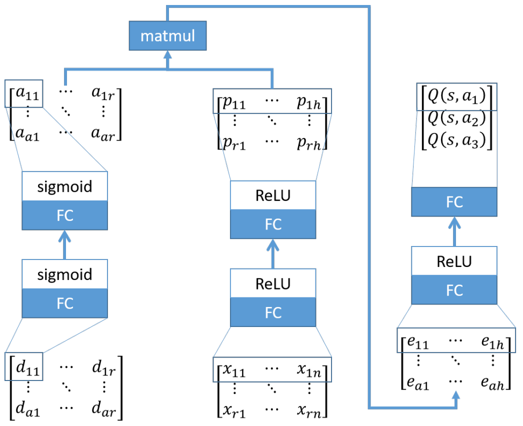

Figure 2: Overview of the function approximation.

from resource $\rho_j \in \mathcal{P}$ to the edge belonging to the i^{th} action. To transfer the distance matrix to a similarity matrix A, we use an MLP f^1 (sigmoid activation) with one input and output dimension, such that $a_{ij} := f^1(d_{ij}, \theta_s)$ for all entries a_{ij} in A. This gives the agent the chance to learn which distances are important. Other possibilities, e.g. to exponentiate the negative (scaled) distance work as well. However, the results of the MLP performed better. Next, $\hat{A}$ is computed by normalizing A such that each row sums up to one. Finally, we can apply the convolution to compute an encoding for each action E by the matrix multiplication $E := \hat{A} \cdot P$.

Action prediction. Finally, we apply another MLP with one output dimension f^1 (ReLU activation after the hidden layer) to every row of E in order to compute the Q-value of every action:

$$Q(s, a_i) := f^1(E_i, \theta_Q), \tag{6}$$

where θ_Q is the parameter set of the MLP and state s produces observation X.

Transferability. To transfer this function approximation to another area, only the graph dependent distance matrix D has to be adapted. In our experiments, we empirically show that transferring to another graph and resource set performs well.

8 Reference Approaches

We compare our solution to the current state-of-the-art TOPs methods [Shao *et al.*, 2017].

Greedy. The *Greedy* approach assumes an exponentially distributed dwelling time of cars being in violation. This approach chooses the next parking spot such that the probability of reaching the next car before its departure is maximized. We choose the next resource to visit with respect to

$$\arg\max_{\rho \in \mathcal{P}_v} exp\left(-\frac{\vartheta_\rho + {}^{d_\rho}/_{V}}{\kappa}\right). \tag{7}$$

$\mathcal{P}_v \subseteq \mathcal{P}$ is the set of all resources currently in violation, ϑ_ρ is the time window since the resource ρ became violated, V is the assumed walking speed and κ is a parameter affecting the average dwelling time [Shao *et al.*, 2017]. Since the exponential function is monotone, it can be omitted. Although this policy is very short-sighted, the computational effort is low and the resulting policy is still valuable.

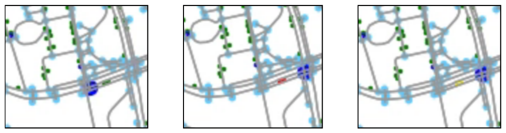

Figure 3: Example where the agent (big blue dot) learned to wait for resources becoming collectible (red) soon. In this simulation, the agent waited multiple time steps (left) until the resources became collectible (middle) and then it collected all resources (right). Light blue dots denote nodes of the graph.

ACO. Our second reference approach uses Ant-Colony-Optimization (*ACO*) techniques [Shao *et al.*, 2017] to approximate a time-varying Travelling Salesman Problem (TSP). In doing so, the approach assumes the same exponential distribution as the Greedy (Formula 7). After every state change, multiple virtual “ants” successively visit resources of $\mathcal{P}_v$. The ants choose the next resource according to present pheromones and the probability that the resource will be in violation at arrival. Since at some point in time, the probabilities that resources are still in violation becomes negligible, the current run can be abandoned before the ant visited all resources.

Note that both approaches ignore some uncertainties and hence, a new solution needs to be re-computed after every non-assumed state change. Therefore both methods can be considered as replanning approaches [Little *et al.*, 2007]. Defining the SRC task as an MDP has a vast advantage compared to the problem definition of [Shao *et al.*, 2017]. MDPs are treating the stochastic future by design, i.e. future appearances of collectible resources are estimated by the policy. The agent can learn to “wait” for close resources becoming collectible. An excerpt of a simulation where the agent waited for resources is illustrated in Figure 3.

9 Experiments

To prevent overfitting, we split the parking event dataset into three sets, training, validation, and test. The segmentation is based on the following principle. If the remainder of the day’s number of the year divided by 13, is zero, the day is added to the test set (28 days). If the rest is one, the day is added to the validation set (27 days). All other days are in the training set. We trained our approach on various GTX/RTX GPU computing machines. The presented results are the best with respect to the validation results after tuning the hyper-parameter (e.g. learning/exploration rate, batch size, hidden neurons). As the ACO algorithm plans on execution time, we assigned a maximum computation time available (1 and 0.1 seconds) for each decision (single core Intel i7-3770 3.40GHz), as we do not think a user would accept a longer waiting time than 1 second plus networking delay. We assumed equal dwelling time κ for all parking spots and we tried different abandon times and chose the parameters according to the validation results.

Results. Table 1 depicts the experimental results for the areas Docklands, Queensberry and Downtown. The values correspond to the average amount of tickets issued per day.

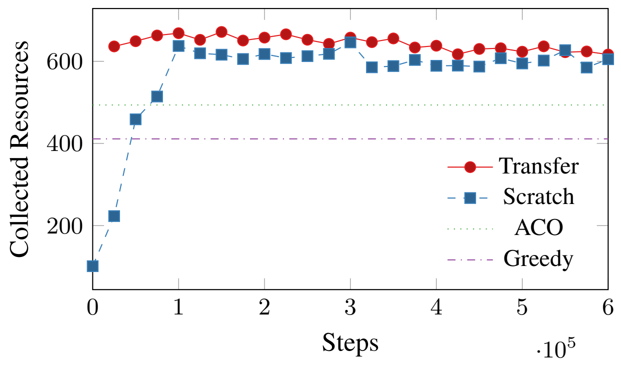

Figure 4: Validation performance during training in Downtown.

Reference approaches. One can see that in all settings the Greedy approach is outperformed by the ACO. The ACO algorithm is slightly stronger than our approaches (SDDQN) in the Queensberry and Docklands setting. This can be explained by the fact that the distances between parking spots are not very large, and hence the stochastic component is not as important as in larger areas, where our solution outperforms both Greedy and ACO significantly (see Figure 4).

Function approximation. To illustrate the benefit of the proposed function approximation, we optimized a standard MLP with the same (flattened) observation (see SDDQN (MLP) in Table 1, Queensberry). It turns out that the MLP performs considerably worse although it is theoretically more powerful since it could learn the correlations between all resources. Nevertheless, our FA shares the weights along all resources. Hence, it is easy to infer experience from one resource to all others. The MLP needs sufficient experiences for each resource. Moreover, the amount of samples needed to learn correlations between resources is high. Thus, the MLP had no chance to significantly improve within the provided period ($600,000$ steps in the environment). Even if the MLP would eventually learn, our FA is much more sample efficient.

Horizon. Even though, our objective is to optimize the total amount of parking tickets issued during the officer's working hours of the officer (hence to optimize $\gamma = 1$), it turns out to be difficult for the agent to learn expectations of events for the far future due to high variance. In particular, the parking states at the beginning of the working day are not very correlated with the parking situation at the end of the day. Consequently, the task is made numerically difficult with $\gamma = 1$. At the same time, it is most important that resources in the near future are not omitted because time periods with limited tickets usually cannot be compensated at a later point in time. This observation is confirmed in our experiments (Table 1). The basic DoubleDQN (DDQN $\gamma = 1$) performs significantly worse than our SMDP-based version on all γ values. Furthermore, one can observe that when selecting too small discount factors, e.g. half-life period of 600 seconds (10 minutes), and too large discount factors, e.g. half-life period of 3600/4800 seconds (1/1.5 hours) the overall performance decreases. Nevertheless, this hyper-parameter still did not react very sensitively.

Area	Algorithm	Val	Test
Queensberry	Greedy	161.7	167.3
	ACO (0.1 seconds)	186.2	190.1
	ACO (1 second)	190.9	196.2
	SDDQN (MLP)	83.6	81.2
	DDQN ($\gamma = 1$)	173.0	167.6
	SDDQN ($\hat{\gamma} = (\frac{1}{2})^{\frac{1}{600}}$)	187.6	184.9
	SDDQN ($\hat{\gamma} = (\frac{1}{2})^{\frac{1}{1200}}$)	189.7	187.0
	SDDQN ($\hat{\gamma} = (\frac{1}{2})^{\frac{1}{2400}}$)	188.6	186.8
	SDDQN ($\hat{\gamma} = (\frac{1}{2})^{\frac{1}{3600}}$)	185.6	181.5
	SDDQN ($\hat{\gamma} = (\frac{1}{2})^{\frac{1}{4800}}$)	185.0	184.5
Docklands	Greedy	223.5	222.4
	ACO (0.1 seconds)	255.6	256.2
	ACO (1 second)	270.1	268.9
	SDDQN	258.2	261.0
Downtown	Greedy	411.1	407.8
	ACO (0.1 seconds)	397.4	412.5
	ACO (1 second)	484.1	497.4
	SDDQN (Scratch)	645.9	643.7
	SDDQN (transfer)	660.3	660.1
	SDDQN (transfer train)	671.4	676.9

Table 1: Summary of the experimental results.

Transfer. We empirically show transferability by applying the weights trained for the Docklands environment (Docklands SDDQN, Table 1) to the Downtown setting. Without any training (SSDQN (transfer)) at the Downtown environment, our approach already outperforms the baseline algorithms (c.f. Table 1). After training, the agent is able to further improve (SSDQN (transfer train)). See also Figure 4.

10 Discussion

We formulate the Stochastic Resource Collection (SRC) task as a discrete time Semi-Markov Decision Process (SMDP) and propose to solve it with an adapted DQN based reinforcement learning approach on a higher level action abstraction. Furthermore, we present a function approximation that shares parameters along all resources and actions (destinations) enabling a transfer to previously unseen areas without further training. In future work, we plan to adopt our approach to multi-agent settings and adapt it to the TDP setting. Furthermore, due to the SMDP formulation, it is suggestive to make use of the options framework. Another research direction could be to replace the graph convolution-like part of the function approximation with an attention mechanism. One bottleneck of our function approximation is the distance matrix D. To make it applicable to very large graphs, pruning far off resources with a sparse matrix representation can be considered.

Acknowledgements

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

[Alabbasi *et al.*, 2019] Abubakr Alabbasi, Arnob Ghosh, and Vaneet Aggarwal. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *arXiv preprint arXiv:1903.03882*, 2019.

[Alshamsi *et al.*, 2009] Aamena Alshamsi, Sherief Abdallah, and Iyad Rahwan. Multiagent self-organization for a taxi dispatch system. In *8th AAMAS, Hungary*, 2009.

[Baird, 1995] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*. Elsevier, 1995.

[Barto and Mahadevan, 2003] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 2003.

[Bellman *et al.*, 1957] R. Bellman, R.E. Bellman, and Rand Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[Bradtke and Duff, 1995] Steven J Bradtke and Michael O Duff. Reinforcement learning methods for continuous-time markov decision problems. In *NIPS*, 1995.

[Crites and Barto, 1998] Robert H Crites and Andrew G Barto. Elevator group control using multiple reinforcement learning agents. *Machine learning*, 1998.

[Gendreau *et al.*, 1996] Michel Gendreau, Gilbert Laporte, and René Séguin. Stochastic vehicle routing. *EJOR*, 1996.

[Godfrey and Powell, 2002] Gregory A Godfrey and Warren B Powell. An adaptive dynamic programming algorithm for dynamic fleet management, i: Single period travel times. *Transportation Science*, 36(1):21–39, 2002.

[Khalil *et al.*, 2017] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *NIPS*, 2017.

[Kim *et al.*, 2019] Joon-Seok Kim, Dieter Pfoser, and Andreas Züfle. Distance-aware competitive spatiotemporal searching using spatiotemporal resource matrix factorization. In *27th ACM SIGSPATIAL*, pages 624–627, 2019.

[Kool *et al.*, 2018] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[Li *et al.*, 2019] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *WWW*, 2019.

[Lillicrap *et al.*, 2015] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

[Lin *et al.*, 2018] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *24th ACM SIGKDD*, pages 1774–1783. ACM, 2018.

[Little *et al.*, 2007] Iain Little, Sylvie Thiebaux, et al. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*, 2007.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *preprint arXiv:1312.5602*, 2013.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 2016.

[Schaul *et al.*, 2015] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[Schmoll and Schubert, 2018] Sebastian Schmoll and Matthias Schubert. Dynamic resource routing using real-time dynamic programming. In *27th IJCAI 2018, Stockholm, Sweden*, pages 4822–4828, 2018.

[Schmoll *et al.*, 2019] Sebastian Schmoll, Sabrina Friedl, and Matthias Schubert. Scaling the dynamic resource routing problem. In *16th SSTD, Vienna, Austria*, 2019.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[Shao *et al.*, 2017] Wei Shao, Flora D Salim, Tao Gu, Ngoc-Thanh Dinh, and Jeffrey Chan. Travelling officer problem: Managing car parking violations efficiently using sensor data. *IEEE Internet of Things Journal*, 2017.

[Shao *et al.*, 2019] Wei Shao, Flora D. Salim, Jeffrey Chan, Sean Morrison, and Fabio Zambetta. Approximating optimisation solutions for travelling officer problem with customised deep learning network. *CoRR*, 2019.

[Solomon and Desrosiers, 1988] Marius M Solomon and Jacques Desrosiers. Survey paper—time window constrained routing and scheduling problems. *Transportation science*, 22(1):1–13, 1988.

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

[Tang *et al.*, 2019] Xiaocheng Tang, Zhiwei (Tony) Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. A deep value-network based approach for multi-driver order dispatching. In *25th ACM SIGKDD, KDD 2019, Anchorage, USA*, pages 1780–1790, 2019.

[Van Hasselt *et al.*, 2016] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *13th AAAI*, 2016.

[Vinyals *et al.*, 2015] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[Watkins, 1989] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[Xu *et al.*, 2018] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *24th ACM SIGKDD*, pages 905–913, 2018.