Reinforcement Learning for Multi-Agent Stochastic Resource Collection

Niklas Strauß[✉][0000−0002−8083−7323], David Winkel[0000−0001−8829−0863], Max Berrendorf[0000−0001−9724−4009], and Matthias Schubert[0000−0002−6566−6343]

LMU Munich
{strauss,winkel,berrendorf,schubert}@dbs.ifi.lmu.de

Abstract. Stochastic Resource Collection (SRC) describes tasks where an agent tries to collect a maximal amount of dynamic resources while navigating through a road network. An instance of SRC is the traveling officer problem (TOP), where a parking officer tries to maximize the number of fined parking violations. In contrast to vehicular routing problems, in SRC tasks, resources might appear and disappear by an unknown stochastic process, and thus, the task is inherently more dynamic. In most applications of SRC, such as TOP, covering realistic scenarios requires more than one agent. However, directly applying multi-agent approaches to SRC yields challenges considering temporal abstractions and inter-agent coordination. In this paper, we propose a novel multi-agent reinforcement learning method for the task of Multi-Agent Stochastic Resource Collection (MASRC). To this end, we formalize MASRC as a Semi-Markov Game which allows the use of temporal abstraction and asynchronous actions by various agents. In addition, we propose a novel architecture trained with independent learning, which integrates the information about collaborating agents and allows us to take advantage of temporal abstractions. Our agents are evaluated on the multiple traveling officer problem, an instance of MASRC where multiple officers try to maximize the number of fined parking violations. Our simulation environment is based on real-world sensor data. Results demonstrate that our proposed agent can beat various state-of-the-art approaches.

Keywords: Multi-Agent RL · Navigation · Deep RL

1 Introduction

In many sequential planning tasks, agents travel on a transportation network, like road or public transportation networks, to reach certain points of interest (POIs) to earn rewards. One way to differentiate these tasks is according to the time intervals for which POIs grant rewards and whether these intervals are known to the agents. For example, for the traveling salesman and the basic vehicular routing problem (VRP), reaching POIs grants rewards regardless of the time they are visited. In more sophisticated tasks such as windowed VRPs [11], POIs only grant rewards during given time windows that are known to the agent. In contrast, in applications like taxi dispatching and ride-sharing, the agent does not

know in advance at which time intervals rewards can be earned. Thus, policies try to guide the agents into areas where collecting rewards is more likely, i.e., passengers might show up.

The task of Stochastic Resource Collection (SRC) [21] assumes that resources have fixed locations and change their availability based on an unknown random process. Thus, the agent observes currently collectible resources and can try to reach these before the resources are not collectible anymore. An instance of the SRC task is the TOP [22] in which a parking officer is guided to fine a maximal amount of parking offenders. The setting is based on the assumption that information about parking sensors is available from sensors registering the duration of parking events. As offenders might leave before the officer arrives, not all resources remain collectible, and thus, agents have to consider the chance of reaching resources in time. [21] model SRCs as Semi-Markov Decision Processes (SMDP) and propose an action space that lets the agent travel to any resource location on a pre-computed shortest path. To find effective policies maximizing the number of collected resources in a given time interval, a reinforcement learning (RL) algorithm based on deep Q-Networks (DQN) is proposed. Though the proposed method learns successful policies for single agents, it often requires more than one agent to handle sufficiently large areas. Thus, [18] propose a multi-agent heuristics for guiding multiple officers in a larger area. As RL methods already showed better performance than known heuristic methods in the single-agent case, it makes sense to examine multi agent reinforcement learning (MARL) methods to improve policies. However, known MARL approaches usually are not designed for Semi-Markov models where agents' actions require varying amounts of time. In addition, they often require mechanisms that counter the problem of the size of the joint action space, which grows exponentially with the number of agents, and the credit assignment problem when using joint rewards. Though there are several methods to counter each of these problems, most of them do not consider the properties of the MASRC environments with asynchronous agent actions in a Semi-Markov environment.

In this paper, we formalize MASRC as a selfish Semi-Markov Game (SMG). We adapt the action space of [21] to let each agent target any resource in the network. Thus, agents generally terminate their actions in varying time steps. We propose a selfish formulation where each agent optimizes its own individual rewards. We argue that a group of independent agents still optimizes the sum of collected resources sufficiently well as the agents learn that evading other agents decreases the chances of another agent collecting close-by resources. We empirically verify our reward design by comparing it to joint rewards. To approximate Q-values, we propose a neural network architecture that processes information about resources, agents, actions, and the relation between them. To combine these types of information, we employ attention and graph neural network mechanisms. This way, our agent can estimate the likelihood of reaching a collectible resource before it becomes uncollectible or another agent reaches the resource first. Furthermore, our resource embedding considers the spatial closeness of additional collectible resources to make actions moving the agent into a

region with multiple collectible resources more attractive. To evaluate our new approach, we developed a multi-agent simulation based on real-world parking data from the city of Melbourne. Our experiments demonstrate superior performance compared to several baselines [18] and (adaptions of) state-of-the-art approaches [21,1,9]. We compare our methods with heuristic methods proposed in [18], an adaption of the single-agent SRC method from [21] and an architecture proposed for dynamic multi-agent VRP [1] which is based on the well-known single-agent architecture [9]. We evaluated the last benchmark to demonstrates that state-of-the-art solutions for the dynamic VRP do not sufficiently cope with the additional stochasticity of MASRC problems. To further justify the design choices in our architecture, we provide ablations studies. To conclude, we summarize the contributions of our paper as:

- A formulation of the MASRC as a Semi-Markov Game building a solid theoretical foundation for the development of MARL approaches
- A novel architecture for learning rich state representations for MARL
- A scalable simulation environment for the multi-agent traveling officer problem (MTOP) problem based on real-world data

2 Related Work

In this section, we review work on related tasks routing an agent through spatial environments to collect rewards. In addition, we will discuss general multi agent reinforcement learning approaches.

2.1 Stochastic Resource Collection

One of the most recognized routing tasks in the AI community is the *vehicular routing problem (VRP)* where a group of agents needs to visit a set of customer locations in an efficient way. There exist various variations of the VRP [4] and some of them include the appearance of new customers during the day [1]. In contrast to SRC, the setting does not include customers disappearing after an unknown time interval. This is a decisive difference as it makes the reward of an action uncertain. In recent years, several approaches have been developed to solve the vehicular routing problem or some of its variations using DRL [1,9,16,17]. MARDAM [1] is an actor-critic RL-agent - based on [9] - designed to solve VRP with multiple agents using attention mechanisms. While state and action spaces of dynamic VRP and MASRC can be considered as very similar, the behavior of the environment is not. To demonstrate these differences, we compare to an agent using the architecture of [1] in our experiments.

There exist various papers on *multi-agent taxi dispatching* [8,12,13,28,32] which can be formulated as a MASRC task. However, in most settings there are significant differences to MASRC as the resources are usually not claimed at arrival. Instead, customers are assigned to close-by taxis the moment the guest publishes a request to the dispatcher. Thus, reaching the guest in time is

usually not considered. Furthermore, to the best of our knowledge, only a single approach works directly on the road network [8]. All other approaches work on grid abstractions which are too coarse for MASRC. Finally, taxi dispatching tasks usually involve large and time variant sets of agents. To conclude, known solutions to taxi dispatching are not applicable to solve MASRC.

The *traveling officer problem (TOP)*, first described by [22] is an instance of SRC. In [21], the authors propose an Semi-Markov RL-based agent to solve the single-agent TOP task and name other tasks that can be formulated as SRC. Later on, the authors of [18] study the MTOP. They propose a population-based encoding, which can be solved using various heuristics for optimization problems like cuckoo search or genetic algorithms. Additionally, they propose a simple greedy baseline that assigns idling officers to the resource in violation using "first-come-first-serve". Competition between officers is handled by assigning a collectible resource to the officer with the highest probability that the resource is still in violation when the officer arrives.

2.2 Multi-Agent Reinforcement Learning

After reviewing solutions to similar tasks, we will now discuss general multi agent reinforcement learning (MARL) approaches w.r.t. their suitability for training on MASRC environments. In MARL, a group of agents shares the same environment they interact with. There are various challenges in MARL: the non-stationarity of the environment from the perspective of an individual agent, the exponentially increasing joint action space, the coordination between agents, and the credit assignment problem. A plethora of different approaches to tackle these challenges exists [6] and we will give a brief overview of the most important MARL approaches in the following.

Joint action learners reduce the multi-agent problem to a single-agent problem by utilizing a single centralized controller that directly selects a joint action. While joint action learners can naturally handle coordination and avoid the non-stationarity, in practice, these approaches are often infeasible because of the exponential growth of the joint action space w.r.t. the number of agents [7].

On the opposite site, we can use multiple independent learners [26]. The agents interact in parallel in a shared environment using a single-agent RL algorithm. In many cases, it has been shown that independent learners can yield strong performance while allowing for efficient training. However, in some settings, independent learners can suffer from the non-stationarity of the environment induced by simultaneously learning and exploring agents.

In recent years, approaches have been developed that utilize centralized training and decentralized execution (CLDE). In [24], the authors presented VDN that decomposes the joint action-value function as a sum of the individual agents' Q-function values obtained solely from the agents' local observation. The authors of [19] propose QMIX, a method that extends VDN by learning a non-linear monotonic combination of the individual Q-functions, which allows representing a larger class of problems. The authors of [3] propose a counterfactual multi-agent actor-critic method (COMA) that uses a centralized critic that allows

estimating how the action of a single agent affects the global reward in order to address the credit assignment problem.

Another way to tackle the problem of coordination between agents is to facilitate communication between the agents. CommNet [23] is a prominent approach that learns a differentiable communication model between the agents. Both CLDE and communication-based approaches suffer from the credit-assignment problem, which we mitigate through our individual reward design.

A drawback of the named approaches when applied to MASRC is that these algorithms do not consider *temporal abstractions*, i.e., actions with varying duration. The application of temporal abstraction to CLDE requires the modification of the problem in a way that the decision epochs are synchronized or experience needs to be trimmed [27]. This way of training is inefficient as it exponentially increases the number of decision epochs with respect to the number of agents. The authors of [2,14,5,20,27] investigate temporal abstraction in multi-agent settings. The authors of [20] first introduce different termination schemes for actions with different temporal abstractions that are executed in parallel. [14] propose independent learners to efficiently handle the asynchronous termination setting, [27] adapt CommNet and QMIX to a setting with temporal abstraction, while [2] propose a version of COMA in decentralized settings with temporal abstractions. Let us note that some of these approaches, like COMA, QMIX, or CommNet, can be adapted to train our function approximation and thus, can be applied to MASRC. We experimented with these approaches but could not observe any convincing benefit for solving MASRC. In addition, the use of those methods tries to learn complex coordination schemes between agents. However, in MASRC agents basically cannot directly support each other as the only action impacting other agents is collecting resources.

3 Problem Formulation

We consider the problem of MASRC, where n agents try to maximize the collection of resources in a road network $G = (V, E, C)$, where V is a set of nodes, E denotes a set of edges and $C : E \to \mathbb{R}^+$ are the corresponding travel costs. Each resource $p \in P$ is located on an edge $e \in E$ in the road network. Whether a resource p is collectible can be observed by the agents but might change over time. The state changes of resources follow an unknown stochastic process. Whenever an agent passes a collectible resource, the resource is collected by the agent.

Formally, we model the MASRC problem as a Semi-Markov Game (SMG) $\langle I, S, \mathbf{A}, P, R, \gamma \rangle$, where I is a set of agents indexed by $1, \ldots, n$, S is the set of states, $\mathbf{A}$ denotes the joint action space, P is the transition probability functions, R denotes the reward functions of the individual agents, and γ is the discount factor.

Agent: A set of n agents moving in road network and collecting resources.

State: $\mathbf{s}_t \in S$ denotes the global state of the environment at time t. The exact information included in the state depends on the actual instantiation, e.g.,

TOP. Nonetheless, all MASRC tasks share a common structure that can be decomposed into resources, agents, and environment:

- *Resources* characterized by the current status, e.g., availability and position.
- *Agents* defined by their position and ID.
- *Environment* with features such as the time of the day or an indication of holidays.

Action: $\mathbf{a}_t \in \mathbf{A} = A_1 \times \ldots \times A_n$: is the joint action at time t. Following the single-agent formulation of [21], we define the individual action space A_i of an agent to correspond to the set of edges E, i.e., the agent will travel on the shortest path to the corresponding edge. This allows to focus on the MASRC task itself rather than solving the routing problem, where high-performance deterministic algorithms are available. Therefore, the individual actions have varying duration, depending on the agent's position and target location. As a result, agents may have to asynchronously select actions at different decision times. Between those decision times, agents continue to their target. Formally, we can reduce this to a synchronous setting, and thus the given joint action space, by introducing a special "continue" action, as described in [14].

Reward: Each agent i has an independent reward function $R_i \in R$, where $R : (S \times \mathbf{A}) \to \mathbb{R}$. Each agent i independently tries to maximize its own expected discounted return $\mathbb{E}\left[\sum_{j=0}^{\infty} \gamma^j r_{i,t+j}\right]$. Each agent's individual reward function corresponds to the resources collected by the agent itself. The reward is incremented by 1 for each collected resource. A resource is collected when an agent passes a collectible resource.

State Transition Probability: With

$$(\mathbf{s}_{t+1}, \tau \mid \mathbf{s}_t, \mathbf{a}_t) : (S \times \mathbb{R}^+ \times \mathbf{A} \times S) \to [0, 1] \subset \mathbb{R} \tag{1}$$

we denote the probability of transitioning to the state $\mathbf{s}_{t+1}$ from the current state $\mathbf{s}_t$ by taking the joint action $\mathbf{a}_t$. Although, some effects of an action are deterministic (e.g., the positions of the agents), the state changes of resources are uncertain and the exact dynamics are unknown. Unlike in a Markov Game, in a SMG, we additionally sample the number of elapsed time-steps τ of the action $\mathbf{a}_t$. The smallest feasible temporal abstraction is the greatest common divisor of all edge travel times. The duration is determined by the individual action $a_i \in \mathbf{a}$ with the shortest duration, which is a multiple of the smallest feasible duration. As a result, an agent receives a time-discounted reward $\zeta = \sum_{j=0}^{\tau-1} \gamma^j r_{t+j+1}$.

4 Method

In this section, we introduce our novel multi-agent RL agent. At first, we provide some insight into our reward design. Secondly, we present our training procedure that is based on independent DQN [25]. After that, we describe the inputs to our architecture and name the particular features for our evaluation on the MTOP task. Finally, we introduce our novel function approximator for the MASRC problem that facilitates coordination between the agents.

4.1 Individual Rewards

In general, the goal of MASRC is to maximize the expected joint reward $\mathbb{E}\left[\sum_{i=0}^{|I|}\sum_{j=0}^{\infty}\gamma^j r_{i,t+j}\right]$. However, we decided to use individual rewards to avert the credit assignment problem, which leads to a Markov Game where agents act selfishly. In literature, the impact of such selfish behavior is commonly denoted as the "price of anarchy" [10]. In the context of MASRC, we argue that the price of anarchy is likely to be very low and outweighed by the benefits of having a reward function that allows the agents to assess the impact of their actions more directly and thus mitigates the credit assignment problem. This is because in MASRC helping other agents directly is not possible. Therefore, coordination boils down to not getting in the way of other agents. There might be cases where a joint reward might lead to policies where particular agents would target far-off resources decreasing their own but increasing the sum of collected resources. However, we observed in our experiments that these cases are rare. We provide an empirical evaluation of our reward design choice compared to joint rewards in Section 6.3.

4.2 Training

Independent DQN [25] combines independent learners [26] and DQN [15]. To speed up learning, we share the network parameters between agents and distinguish them by their IDs [31]. Independent learning provides a natural way to handle settings with asynchronous termination [14]. In independent learning, each agent treats the other agents as part of the environment. However, this may lead to sub-optimal coordination between the agents. To mitigate this problem, we introduce an architecture that allows each agent to efficiently reason about the intents of other agents. We utilize a DoubleDQN [29] adapted to the Semi-Markov setting. We update the network parameters with respect to a batch of transitions collected from all agents by minimizing the following loss function:

$$\mathcal{L}(\Theta) = \mathbb{E}_{\mathbf{s}_t, a_t, \tau, r_{t:t+\tau-1}, \mathbf{s}_{t+\tau}}\left[loss(y_t, Q(\mathbf{s}_t, a_t; \Theta))\right] \tag{2}$$

where $y_t = \sum_{j=0}^{\tau-1}\gamma^j r_{t+j} + \gamma^\tau Q(s_{t+\tau}, a'_{t+\tau}; \Theta')$. The action $a'_{t+\tau}$ is the optimal action w.r.t to Θ, i.e., $a'_{t+\tau} = \text{argmax}_{a_{t+\tau}\in A(s_{t+\tau})} Q(s_{t+\tau}, a_{t+\tau}; \Theta)$. Θ denotes the parameters of the behavior Q network and Θ' denotes the parameters of the frozen target Q network which are periodically copied from Θ. To improve clarity, we omitted the indices indicating the individual agents. We use a smooth L1 loss.[1].

4.3 Input Views

In the following, we will briefly describe the inputs to our function approximation and name the particular features for our evaluation on the MTOP task.

[1] cf. `https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html`

Resource Features We encode each resource from the perspective of each individual agent separately. To this end, we add features describing the relation of the agent to the resource, e.g., the distance or arrival time. This results in n times different views of each resource. The resource view for the MTOP contains a one-hot-encoding of the resource's current status, i.e., free, occupied, in violation, or fined. Additionally, we provide a flag that indicates whether a parked car would be in violation if it remains parked and the officer would directly go there. Finally, we add the current time of the day, walking time, agent arrival time, and distance to the resource. All these features are normalized. 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. A score greater than zero indicates a violation. Finally, we add the normalized coordinates of the resource's position.

Agent Features For MTOP, it consists of a one-hot encoding of the agents' ID, the normalized coordinates of its current position and target, as well as the normalized walking time and distance to its target.

Spatial Relation To capture the spatial interaction between the resources, we create a distance matrix for each agent. There is one row in the matrix for each action consisting of the network distance of the action target to each resource, the distance between the agent, and the action target to each row.

4.4 Architecture

An effective policy in MASRC requires an agent to consider the complex interaction between resources, actions, and other agents to estimate the likelihood of reaching a *collectible* resource. To capture those dependencies, our novel architecture first encodes the action-level intents of each agent using the resources and their spatial relationship solely from the perspective of each individual agent, i.e., ignoring the other agents. We call this module the *Shared Action Encoder*. After that we continue by combining the perspective of the current agent with the action-level intents of the other agents using multi-head attention in the *Intent Combination Module*. This allows an agent to asses the likelihood that another agent catches collectible resources first. While the inputs are different, the parameters of all networks are shared between all agents.

Shared Action Encoder In the context of SRC, the value of an action, i.e., the likelihood of reaching resources in time, depends largely on the state of resources near the target [21]. We argue that in a multi-agent setting, the simple distance weighting from [21] is not expressive enough to capture the complex dependency of an action's value on, e.g., the uncertainty of reaching the resource in time. Thus, we propose an extended *Shared Action Encoder* to calculate agent-specific action embeddings, based upon the agent's features, and resources' features, as well as the distances. We provide the pseudo-code roughly following PyTorch style in Fig. 2, and show an overview in Fig. 1.We begin by transforming the agent's features with an MLP (cf. line 3). Next, we calculate unnormalized agent-specific resource to action relevance scores combining information from the agent

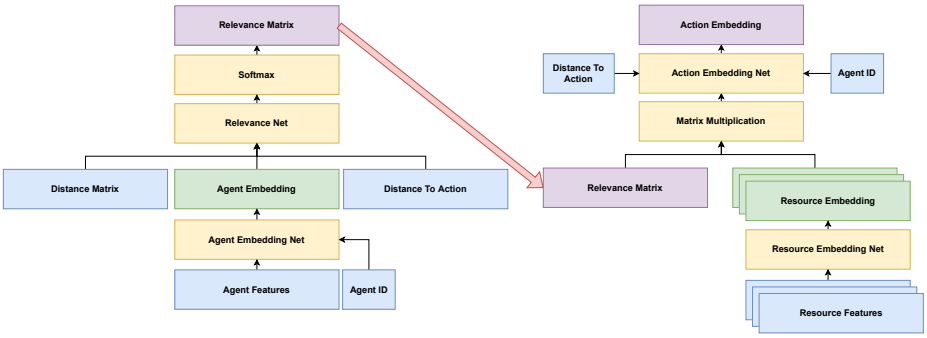

Fig. 1: Conceptual overview of the *Shared Action Encoder*. This module creates a rich representation for each action based on the resource states from the perspective of an individual agent. The module captures the spatial relationship of resources around each action's target using a graph neural network mechanism. Networks and operations are colored yellow, the output of the module and crucial intermediate representations are purple, while blue denotes input features.

```
def sea(feat_ag, feat_res, i_ag, dist, dist_ag2ac):
    """Shared action encoder for a single agent."""
    x_ag        = mlp1(feat_ag)
    # shape: (dim_ag,)
    rel_act_res = mlp2(cat(broadcast([
        x_ag[None],
        dist_ag2ac[None],
        dist,
    ]), dim=-1))
    # shape: (n_action, n_res)
    rel_act_res = softmax(rel_act_res, dim=-1)
    # shape: (n_action, n_res)
    x_res       = mlp3(feat_res)
    # shape: (n_res, dim_res)
    x_act       = rel_act_res @ x_res
    # shape: (n_act, dim_res)
    return mlp4(cat(x_act, i_ag, dist_ag2ac))
```

Fig. 2: Pseudocode for the *Shared Action Encoder* following PyTorch style. `feat_ag` denotes the agent's features, `feat_res` the (agent-specific) resource features, `i_ag` the agent's ID, and `dist` the action-resource distance matrix, and `dist_ag2ac` the distance from the current agent to all actions (which are target edges). `mlp1` to `mlp4` are separate MLPs.

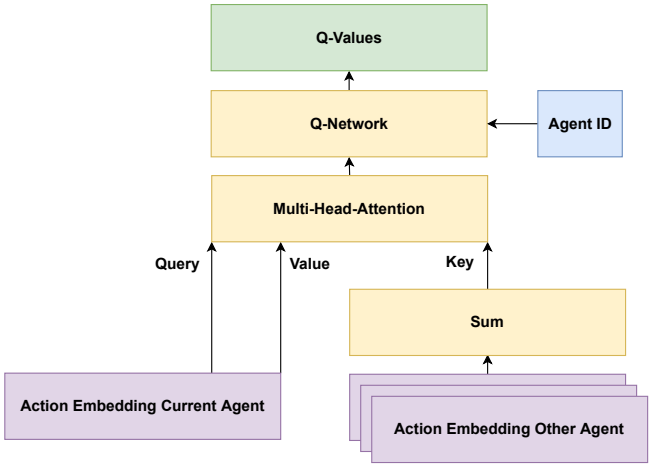

Fig. 3: In the *Intent Combination Module*, we enrich the action embedding of a single agent with information about the other agents' actions using multi-head attention. Afterwards, we reduce the enriched action representations to a Q-value for every action with an MLP. Networks and operations are colored yellow, inputs coming from the *Shared Action Encoder* are purple, and blue denotes input features.

representation, the distance from the agent to the action (i.e., edge), and the action-to-resource distance matrix using another MLP (cf. lines 5-9). These relevance scores are subsequently normalized using the softmax operator (cf. line 11). The resource features are first transformed by an MLP, before we use the previously computed relevance scores for aggregating them per action (cf. line 13-15). A final MLP combines this information with distance to the agent as well as the agent ID (cf. line 17). In the following, we denote the result of this component as $\mathbf{E}$.

Intent Combination Module Information about the other agents' intents is crucial in multi-agent settings - thus, we propose an attention-based mechanism to update an agent's action representations by considering the ones of other agents. Let $\mathbf{E}_i$ denote the output of the shared action encoder for agent i. For scalability, we first aggregate the latent actions of all other agents $\overline{\mathbf{E}}_i := \sum_{j \in I \setminus \{i\}} \mathbf{E}_j$. Next, apply a multi-head attention mechanism [30] with $\mathbf{E}_i$ as query and value and $\overline{\mathbf{E}}_i$ as key. Finally, we reduce every row of the result, corresponding to the co-agents-aware action representation, to a single Q-value using an MLP. This MLP also receives the agent ID as an additional input to allow diversification of the agents.

Area	Nodes	Edges	Resources	Edges with Resources
Docklands	1,435	4,307	487	166
Queensberry	1,711	5,356	639	177
Downtown	6,806	21,369	1,481	493

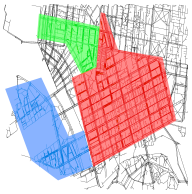

Fig. 4: Description and illustration of the different areas used in our evaluation: Docklands (blue), Downtown (red), and Queensberry (green). Notice that typically only a small fraction of edges contains resources and there can be more than one resource per edge.

5 Simulator Design

To the best of our knowledge, there is no publicly available simulation for MTOP. To enable effective training of reinforcement learning agents, we implement a simulator that can replay real-world sensor data and parking restrictions, which allows us to simulate as close as possible to the real world. The walking graph, i.e., road network, is extracted from OpenStreetMap[2]. We assign parking spots to the closest edge in the graph. When an agent passes a resource in violation, it will be fined. The time for fining a violation is set to zero in our simulation. The agent collects a reward of $+1$ for every fined resource. All agents start at the same place every day. They work for 12 hours from 7 am to 7 pm. Each agent has a walking speed of 5km/h.

For our evaluation, we use openly available on-street parking sensor data and parking restrictions from the city of Melbourne in 2019 [3]. We divide Melbourne into three areas to study different graph structures and hyperparameter transferability. Details regarding the areas can be found in Fig. 4. Each run was trained using a single GPU on a cluster consisting of RTX A6000 (48GB) and A100 (40GB) GPUs. The code of our simulation and agents is publicly available.[4]

6 Experimental Evaluation

We split the parking event dataset into a training, validation, and test set. Parking follows weekly patterns. To avoid biases introduced through weekdays, we split the dataset as follows: If the remainder of the day in the year divided by 13 is 0, we add the day to the test set. In case the remainder is 1, we add the day to the validation set. The remaining 308 days are added to the training set. An episode is equivalent to a working day. The order of the training days is shuffled. To speed up training, the agent interacts with eight environments in parallel.

The transferability of the hyperparameters across different regions and numbers of agents is important. We tuned the hyperparameters in a single area

[2] `https://www.openstreetmap.org`
[3] `https://data.melbourne.vic.gov.au/browse?tags=parking`
[4] `https://github.com/niklasdbs/masrc`

(Docklands) with two agents. Agents were trained using early stopping. The test results reported are with respect to the best validation results. The full hyperparameter setting can be found in the supplement.

6.1 Baselines

Greedy We modify the greedy baseline from [18] for better performance: Instead of assigning agents to the resource with the earliest violation time, we directly use the catching probability from the the tie-breaking mechanism.

LERK The authors of [18] propose to solve the MTOP by representing it using leader-based-random-key encoding (LERK) and then solve it using various classical heuristic solvers developed for combinatorial issues. One of these heuristics that yielded the best performance was the genetic algorithm, which we have implemented.

MARDAM [1] is an actor-critic RL-agent - based on [9] - designed to solve dynamic-VRP with multiple agents using attention mechanisms. While they propose a method to transform the underlying Markov game into a sequential MDP, this transformation is not possible for MASRC tasks. Therefor, we train their architecture using independent actor-critic. Due to the dynamic state of resources in MASRC tasks, we need to calculate the customer-embeddings (i.e., resource), in every step using a Transformer which is computationally expensive and memory intense. As a result, we are not able to train the agent on full episodes and need to rely on bootstrapping.

SASRC We train the architecture of [21] that has been proposed for the SASRC using independent learning with shared independent learners. We add the agent-id to the final network so that agents can differentiate their behavior. Additionally, we add information about the targets of all agents to the resources, which allows the agent to incorporate information about other agents and thus benefits learning [31].

6.2 Results

As the evaluation metric, we use the average number of violations fined per day. We evaluate in three different areas using two, four, and eight agents. The results in Table 1 show that our proposed approach can surpass the other approaches and baselines in various regions and across different numbers of agents. We can beat MARDAM, a state-of-the-art algorithm designed for multi-agent dynamic-VRP, by a large margin. This underlines that approaches for SRC tasks need to be able to handle the increased stochasticity. Moreover, MARDAM requires a massive amount of GPU memory due to the use of the transformer encoder in large settings like Downtown with eight agents. For this setting, our approach

Table 1: Average number of violations fined per day in Docklands, Queensberry, and Downtown for 2, 4, and 8 agents on the validation and test set.

		2 Agents		4 Agents		8 Agents	
Area	Algorithm	Validation	Test	Validation	Test	Validation	Test
Docklands	Greedy	186.93	192.67	304.32	300.22	442.57	439.33
	LERK	244.78	245.11	328.61	330.56	424.32	418.19
	MARDAM	339.79	336.37	418.96	416.44	482.57	479.78
	SASRC	343.82	304.19	476.07	465.52	551.86	544.63
	OURS	**388.32**	**379.59**	**527.21**	**518.52**	**588.04**	**580.00**
Queensberry	Greedy	180.46	189.15	240.5	250.07	277.54	286.56
	LERK	192.18	198.78	233.86	245.07	260.79	271.67
	MARDAM	225.18	229.85	247.29	255.41	257.04	267.81
	SASRC	222.61	231.41	257.11	266.00	263.79	273.15
	OURS	**244.43**	**255.41**	**271.39**	**281.59**	**284.75**	**294.37**
Downtown	Greedy	138.79	144.63	256.14	257.33	429.93	435.22
	LERK	213.57	219.19	298.32	305.40	430.18	428.19
	MARDAM	340.54	342.37	469.18	471.26	255.82	260.93
	SASRC	425.68	418.48	657.61	658.04	815.68	815.41
	OURS	**495.07**	**494.70**	**710.75**	**713.3**	**866.04**	**867.93**

uses approximately 16 times less GPU memory during training. As a result, the batch size needs to be reduced for those settings, which may impact performance. Additionally, the episode length in MASRC tasks is much longer than in a typical VRP, which makes learning on whole episodes impossible. Furthermore, the experiments show that our approach yields considerably better results than existing heuristic solvers designed for the MTOP, such as LERK, which require intensive computational resources at inference time. While our approach requires several days of training it only needs a few milliseconds at inference time.The authors of LERK [18] state a runtime of 4.67 minutes for making a single decision with seven officers using their fastest approach. This makes the application of their algorithm in real-world settings infeasible.

6.3 Ablation Studies

To assess the individual components' impact on the final performance, we provide several ablation studies. We conduct the ablations with two agents on the validation set in the Docklands area and report the average number of violations fined per day. The results of the ablations can be found in Table 2, where they are sorted decreasingly by performance, i.e., the highest impact is on the right.

We observe that not using an *action embedding network* has the strongest impact on the performance resulting in an 8.3% reduction in performance. Since the reduction is less severe when removing inputs of this network, the effect can be primarily attributed to the additional non-linear transformation after resource aggregation. Switching from individual to joint rewards is next in terms of relevance. We observe that using *joint rewards* performs considerably worse,

Table 2: Ablations performed in the area of Docklands with two agents. We report the average number of caught violations per day. The second row shows the relative performance compared to the base configuration. The values are sorted decreasingly, i.e., the highest impact is on the right.

	OURS	With Other Agents' Target	Without Agent ID	Without Resource Position	Without Agent Embedding Network	Without Distance To Action	Joint Reward	Without Action Embedding Network
absolute	**388.32**	378.54	375.39	370.36	365.25	360.96	359.71	356.18
relative	100.0%	97.5%	96.7%	95.4%	94.1%	93.0%	92.6%	91.7%

leading to a 7.4% reduction in performance,[5] which we attribute to the credit assignment problem. Ignoring the *distance to the action* leads to a reduction of 7.0%. Without this information the agent lacks input to assess the inherent reward uncertainty in far actions. The *agent embedding network* is the next crucial component, with a reduction of 5.9%. Without it, the model cannot utilize the agent features, such as its position. Not having access to the *agent ID* aggravates diversification of agent policies and leads to a performance decrease of 3.3%. Finally, *adding other agents' target information* to the agent-specific views of the resources leads to slightly worse performance of around 2.5%, despite yielding improvements in the SASRC baseline. This indicates that our architecture can already sufficiently incorporate the intents of other agents for effective coordination.

7 Conclusion

In this work, we have formalized Multi-Agent Stochastic Resource Collection (MASRC) as a Semi-Markov Game, providing a solid theoretical framework for the development of new approaches. We further proposed a novel architecture to solve MASRC tasks featuring an innovative intent combination model which permits re-assessment of action representations based on the other agents' action representations. To enable evaluation, we introduced an efficient agent-based simulation for the MTOP task, for which we publish the source code to support the community in future research. Using the simulation, we could demonstrate that our approach is able to beat existing heuristic baselines, adaptions of state-of-the-art single-agent SRC solutions, and approaches for the multi-agent dynamic-VRP in terms of fined violations. On a more fundamental level, our results indicate that existing approaches for multi-agent dynamic-VRP struggle to handle the increased dynamics in MASRC tasks, and thus MASRC requires specialized solutions. In future work, we want to include dynamic travel times.

[5] Notice though that even with joint rewards, our approach is able to beat baselines trained with individual rewards.

Furthermore, we want to investigate the transfer of trained policies between different areas and numbers of agents. Finally, we will research further scaling our approach to very large graphs.

8 Acknowledgments

We thank the City of Melbourne, Australia, for providing the parking datasets used in this paper and Oliver Schrüfer for contributing the implementation of LERK. 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

1. Bono, G., Dibangoye, J.S., Simonin, O., Matignon, L., Pereyron, F.: Solving multi-agent routing problems using deep attention mechanisms. IEEE Trans. Intell. Transp. Syst. **22**(12), 7804–7813 (2020)
2. Chakravorty, J., Ward, N., Roy, J., Chevalier-Boisvert, M., Basu, S., Lupu, A., Precup, D.: Option-critic in cooperative multi-agent systems. arXiv preprint arXiv:1911.12825 (2019)
3. Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S.: Counterfactual multi-agent policy gradients. In: AAAI. vol. 32 (2018)
4. Gendreau, M., Laporte, G., Séguin, R.: Stochastic vehicle routing. Eur. J. Oper. Res. **88**(1), 3–12 (1996)
5. Han, D., Boehmer, W., Wooldridge, M., Rogers, A.: Multi-agent hierarchical reinforcement learning with dynamic termination. In: Pacific Rim Int'l Conf on Artificial Intelligence. pp. 80–92. Springer (2019)
6. Hernandez-Leal, P., Kartal, B., Taylor, M.E.: A survey and critique of multiagent deep reinforcement learning. Auton Agent Multi Agent Syst **33**(6), 750–797 (2019)
7. Hu, J., Wellman, M.P., et al.: Multiagent reinforcement learning: theoretical framework and an algorithm. In: ICML. vol. 98, pp. 242–250. Citeseer (1998)
8. Kim, J., Kim, K.: Optimizing large-scale fleet management on a road network using multi-agent deep reinforcement learning with graph neural network. In: ITSC. pp. 990–995. IEEE (2021)
9. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475 (2018)
10. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Annual symposium on theoretical aspects of computer science. pp. 404–413. Springer (1999)
11. Kumar, S.N., Panneerselvam, R.: A survey on the vehicle routing problem and its variants (2012)
12. Li, M., Qin, Z., Jiao, Y., Yang, Y., Wang, J., Wang, C., Wu, G., Ye, J.: Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In: The world wide web conference. pp. 983–994 (2019)
13. Liu, Z., Li, J., Wu, K.: Context-aware taxi dispatching at city-scale using deep reinforcement learning. IEEE Trans. Intell. Transp. Syst. (2020)
14. Makar, R., Mahadevan, S., Ghavamzadeh, M.: Hierarchical multi-agent reinforcement learning. In: Proc. of the fifth Int'l Conf on Autonomous agents. pp. 246–253 (2001)

15. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
16. Nazari, M., Oroojlooy, A., Snyder, L., Takác, M.: Reinforcement learning for solving the vehicle routing problem. Adv Neural Inf Process Syst **31** (2018)
17. Peng, B., Wang, J., Zhang, Z.: A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In: International Symposium on Intelligence Computation and Applications. pp. 636–650. Springer (2019)
18. Qin, K.K., Shao, W., Ren, Y., Chan, J., Salim, F.D.: Solving multiple travelling officers problem with population-based optimization algorithms. Neural Computing and Applications **32**(16), 12033–12059 (2020)
19. Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., Whiteson, S.: Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: ICML. pp. 4295–4304. PMLR (2018)
20. Rohanimanesh, K., Mahadevan, S.: Learning to take concurrent actions. Adv Neural Inf Process Syst **15** (2002)
21. Schmoll, S., Schubert, M.: Semi-markov reinforcement learning for stochastic resource collection. In: IJCAI. pp. 3349–3355 (2021)
22. Shao, W., Salim, F.D., Gu, T., Dinh, N.T., Chan, J.: Traveling officer problem: Managing car parking violations efficiently using sensor data. IEEE Internet of Things Journal **5**(2), 802–810 (2017)
23. Sukhbaatar, S., Fergus, R., et al.: Learning multiagent communication with backpropagation. Adv Neural Inf Process Syst **29** (2016)
24. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J.Z., Tuyls, K., et al.: Value-decomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296 (2017)
25. Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., Vicente, R.: Multiagent cooperation and competition with deep reinforcement learning. PloS one **12**(4), e0172395 (2017)
26. Tan, M.: Multi-agent reinforcement learning: Independent vs. cooperative agents. In: ICML. pp. 330–337 (1993)
27. Tang, H., Hao, J., Lv, T., Chen, Y., Zhang, Z., Jia, H., Ren, C., Zheng, Y., Meng, Z., Fan, C., et al.: Hierarchical deep multiagent reinforcement learning with temporal abstraction. arXiv preprint arXiv:1809.09332 (2018)
28. Tang, X., Qin, Z., Zhang, F., Wang, Z., Xu, Z., Ma, Y., Zhu, H., Ye, J.: A deep value-network based approach for multi-driver order dispatching. In: Proc. of the 25th ACM SIGKDD. pp. 1780–1790 (2019)
29. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI. vol. 30 (2016)
30. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Adv Neural Inf Process Syst **30** (2017)
31. Zheng, L., Yang, J., Cai, H., Zhou, M., Zhang, W., Wang, J., Yu, Y.: Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In: AAAI. vol. 32 (2018)
32. Zhou, M., Jin, J., Zhang, W., Qin, Z., Jiao, Y., Wang, C., Wu, G., Yu, Y., Ye, J.: Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In: Proc. of the 28th ACM Int'l Conf on Information and Knowledge Management. pp. 2645–2653 (2019)