Online Collective Multiagent Planning by Offline Policy Reuse with Applications to City-Scale Mobility-on-Demand Systems

Wanyuan Wang
Southeast University
Nanjing, China
wywang@seu.edu.cn

Gerong Wu
Southeast University
Nanjing, China
grwu@seu.edu.cn

Weiwei Wu
Southeast University
Nanjing, China
weiweiwu@seu.edu.cn

Yichuan Jiang
Southeast University
Nanjing, China
yjiang@seu.edu.cn

Bo An
Nanyang Technological University
Singapore
boan@ntu.edu.sg

ABSTRACT

The popularity of mobility-on-demand (MoD) systems boosts the need for online collective multiagent planning, where spatially distributed servicing agents are planned to meet dynamically arriving demands. For city-scale MoDs with a population of agents, it is necessary to find a balance between computation time (i.e., real-time) and solution quality (i.e., the number of demands served). Directly using an offline policy can guarantee real-time, but cannot be dynamically adjusted to real agent and demand distributions. On the other hand, search-based online planning methods are adaptive. However, they are computationally expensive and cannot scale up.

In this paper, we propose a principled online multiagent planning method, which reuses and improves the offline policy in an anytime manner. We first model MoDs as a collective Markov Decision Process ($\mathbb{C}$-MDP) where the history collective behavior of agents affects the joint reward. We propose a novel state value function to evaluate the policy, and a gradient ascent (GA) technique to improve the policy. We show that GA-based policy iteration (GA-PI) on local policy can converge. Finally, given real-time information, the offline policy is used as the default plan and GA-PI is used to improve it and generate an online plan. Experimentally, the proposed offline policy reuse method significantly outperforms standard online multiagent planning methods on MoD systems like ride-sharing and security traffic patrolling in terms of computation time and solution quality.

KEYWORDS

Multiagent Planning; Markov Decision Process; Policy Reuse

ACM Reference Format:
Wanyuan Wang, Gerong Wu, Weiwei Wu, Yichuan Jiang, and Bo An. 2022. Online Collective Multiagent Planning by Offline Policy Reuse with Applications to City-Scale Mobility-on-Demand Systems. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

1 INTRODUCTION

Mobility-on-demand (MoD) systems are transforming urban mobility by providing convenient and timely service to demands [2]. Such MoD systems include ride-sharing where vehicles drive to meet passengers' requirements [22] and security traffic patrolling where police officers patrol to respond to emergencies [25]. In MoD systems, a population of agents (e.g., vehicles and police officers) are aggregated and planned to meet customer demands (e.g., passengers and emergencies). Due to the uncertainty in demand arrival, online collective multiagent planning (Online_CMP) has attracted lots of attention [10, 11, 34]. A key characteristic of Online_CMP problems is that agents should be sequentially planned for dispatching and repositioning, and the planning in one period has direct impact on following periods' decision making.

Many existing approaches have attempted to find a balance between computation time (i.e., real-time) and solution quality (i.e., the number of demands served) for Online_CMP. To guarantee real-time, the offline policy can be used as a guide to online planning. For example, using historical data to construct the offline model, efficient offline solutions can be achieved by linear programming (LP) [7, 12, 18, 35]. Directly using the offline policy, however, cannot be adapted to real agent and demand distributions. Multiagent reinforcement learning (MARL) can learn how to take actions by observing the real environment [8, 19, 26–29]. Since the MoD environment will be non-stationary with thousands of agents learning concurrently, MARL methods are not stable and may trap into sub-optimal solutions. Online adaptive methods proceed with a rolling horizon and attempt to search a good planning sequence for the current observation at each period [10, 22]. However, searching is time-consuming, online adaptive methods for multiagent systems often have difficulty in computing policies in real-time with a sufficiently long look-ahead horizon [9].

In general, most existing offline learning and online search policies are separately used, an exception is the recently proposed two-stage offline learning online planning (OLOP) framework [33, 39]. In the offline stage, given the historical data, MARL is employed to learn the state value function. In the online stage, agents are always matched with demands that have larger advantage values. Unfortunately, in the OLOP framework, MARL is used to learn the state value function, which might be sub-optimal and can reduce the online planning efficiency. Moreover, online one-step lookahead planning again traps into the dilemma of serving current real demands or anticipated future demands (i.e., state value).

In this paper, we trade computation time with solution quality by proposing an anytime ***p***olicy ***i***teration (PI) algorithm. We adopt the representation for a collective Markov Decision Process ($\mathbb{C}$-MDP)

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online.

[26, 35] in which the collective behavior of agents affects the joint reward. By examining how local policy of each state influences successive state-action reward, we first design a version of state value function for policy evaluation. We propose a ***g***radient ***a***scent (GA) technique to improve the policy. We show that the GA-based PI (GA-PI) can converge within finite iterations. Given real-time observations, we use the optimal offline policy of $\mathbb{C}$-MDP as the default online plan. GA-PI is used to restart from the current plan and improve it in the new iteration, thereby an efficient online plan can be generated. Finally, we validate our GA-PI algorithm on a synthetic dataset and two real-world datasets on ride-sharing and security traffic patrolling. Experimental results show that 1) GA-PI is guaranteed to solve the $\mathbb{C}$-MDP problem and find offline optimal solutions, 2) GA-PI-based offline policy reuse significantly outperforms the state-of-the-art online planning methods that use offline policy directly or online policy search.

2 RELATED WORK

In this section, we briefly summarize related offline learning, online policy search, policy iteration methods and ridesharing .

Offline Learning Methods. Due to stochastic demand-supply dynamics in MoD systems, offline learning methods can be further grouped into model-based and model-free categories. In the model-based category, the historical data is used to build the model, such as traffic delay and demand distributions. Linear programming [7, 35] and dynamic programming [36] techniques can be used to solve the offline problem. A piecewise linear approximation is presented in [18] for the non-linear reward function. In the model-free category, multi-agent reinforcement learning (MARL), which learns a policy by interacting with the complex MoD systems, has been proposed to find an approximate solution. To allow coordination among agents, context constraints are utilized to align state values [20], and the mean field approximation is used to model local interactions [19]. To achieve trade-off between immediate and future gains, a hierarchical RL is proposed [17], where the centralized manager sets abstract goals and the worker takes actions to satisfy the goals. Centralized training can balance the supply and demand distributions by the KL divergence optimization [41]. The imbalance between demand and supply in future periods can also be integrated for reward design [8, 16]. By capturing the mixture of agents distribution and policy, an expectation maximization-based inference approach is proposed to optimize the policy [26]. However, these offline methods cannot be adaptive to real agent and demand distributions [11, 21].

Online Policy Search Methods. Online planning methods typically consider demands that are revealed incrementally over time and making decisions based on these real demands. A greedy algorithm is proposed to maximize the matching between the observed demands and supplies [40]. To address the myopic inefficiency, Lowalekar et al. [22] propose a multi-stage optimization framework, where future demands can be sampled from historical data. At each decision period, given the current observed and future anticipated demands sampled from the historical data, integer program (IP) can be formulated to search the optimal solution. To mitigate the computation expense, the multi-sample multi-stage IP can be approximated by the Lagrangian dual decomposition [21], network flow average [38], and Monte-Carlo Tree Search [10]. Unfortunately, due to inefficiency of searching in an exponential planning search space, most existing online planning methods have difficulty in computing policies in real-time with a sufficiently long look-ahead horizon.

Online Adaptive Methods by Offline Policy Reuse. Guided by the offline policy $\pi(a|s)$ that determines the probability of taking action a at state s, a straightforward online adaptive method is allocating agents to each demand proportional to π [11, 12]. To plan multiple agents to serve multiple demands, Xu et al. [39] propose a bipartite matching between demands and agents, where the weight of an edge is modeled by the state value learned offline. The state-value function can be further improved periodically in an online manner [33]. However, the MARL-based state value function learned might be sub-optimal and inefficient for guiding online planning. Moreover, online matching again traps in the dilemma of serving current real demands or anticipated future demands. Compared with these most related offline policy reuse methods, our proposed GA-PI method can find optimal solutions on the offline $\mathbb{C}$-MDP, which can be used as an efficient baseline to the online planning.

Policy Iteration Methods. Policy Iteration (PI) lies at the core of RL and many planning methods [31]. The classic PI algorithm repeats consecutive stages of policy evaluation and policy improvement with respect to a value function. For a single agent MDP, Bellman equation [3] is efficient for value function evaluation. However, due to the dependence of reward function on historical collective behaviors of agents, Bellman equation cannot apply to $\mathbb{C}$-MDP. For RL with function approximation [37], policy gradient methods [27] have been widely used to learn and improve the policy parameter. Traditional parameterized policy approximation is only convergent to a locally optimal policy [32]. We overcome such technical difficulties by examining how history collective behaviors influence successive state-action reward and designing a novel state valuation function without parameter approximation.

Ride-sharing. In multi-capacity ride-sharing applications, there are a set of (known) requests to be serviced, and a set of available vehicles. Each vehicle should be allocated to a group of requests with the capacity and travel delay constraints, and the objective is to maximize the number of requests serviced [2, 5]. When requests arrive dynamically, they will be inserted into existing routes in a real-time manner [23, 24]. Multi-capacity ride-sharing falls into the vehicle routing literature that makes the best greedy allocations. However, it does not consider vehicle routing in the context of sequential decision problems. In contrast, our work focuses on sequential vehicle re-positioning that considers the impact of current vehicle-request allocation on future allocations.

3 THE MODEL

In our motivating MoD problems of interest, a population of n agents $Q = \{q_1, q_2, \ldots, q_n\}$ are available. The agents are planned to serve demands in sequence for T periods. In ride-sharing, agents represent vehicles and demands represent passenger orders [11, 12], and in security traffic patrolling, agents represent police officers and demands represent traffic/crime emergencies [25, 30]. Let $V = \{v_1, v_2, \ldots, v_m\}$ be the set of m regions in a city. Different regions might have different demands and their demands vary over periods.

The decision of planning agents to demands at one period has an impact on subsequent periods. MDP is an ideal model for sequential MoD problems. To characterize how the collective behavior of agents affects the joint reward, we adopt a collective MDP ($\mathbb{C}$-MDP) representation [27, 35]. Different from the single agent MDP, the reward function in $\mathbb{C}$-MDP depends on history policy rather than merely on current state and action. Formally, a $\mathbb{C}$-MDP can be described by $\mathcal{M} = \langle S, A, T, R, o, s_0 \rangle$:

- **State.** S is a finite set of spatial-temporal states. Each state $s \in S$ is a tuple (t, v), where t is the current period and v is the current region. Let $t(s)$ and $v(s)$ denote the period and region of state s. Here, we assume that all agents start from a source state s_0. Throughout this paper, the set S includes the source state s_0, unless specified.
- **Action.** A is a finite set of actions. The set of actions available at state $s = (t, v)$, $A(s)$, is the set of regions that can be reached from the region v. For example, the action $a = \longrightarrow v_j$ denotes that the agent is planned to move to the region v_j.
- **Transition.** $T(s, a, s') \in [0, 1]$ is the transition probability of ending up at state s' after taking action a at state s. In reality, due to congestion or traffic signals, there may be stochastic delays that disrupt the mobility. This transition function can be estimated by the Google Map using the daily travel time between regions.
- **Demand Distribution.** $o(s, a) = \{o_0(s, a), o_1(s, a), \cdots\}$ denotes the probability distribution of demand $\langle s, a \rangle$, where $o_k(s, a) \in [0, 1]$ is the probability of k demands $\langle s, a \rangle$ requested at s, and $\sum_k o_k(s, a) = 1$. The demand $\langle s, a \rangle$ can be served by the agent taking action a at state s. For example, in ride-sharing, the demand $\langle s, a \rangle$, where $a = \longrightarrow v_j$ indicates the passenger order that wish to pick up at s and drop off at the region v_j. At state s, once an agent plans to travel to another region v_j, he can serve the passenger order with the same origin-destination type, i.e., starting from $v(s)$ and going for v_j. As typically assumed in the ride-sharing literature [8, 11, 21], this demand distribution can be estimated by the historical demand data, which is often available using GPS traces of the taxi fleet.
- **Reward.** $R(s, a)$ is the immediate reward for taking action a at state s. In MoDs, each demand can only be served by one agent. For example, in ride-sharing, one vehicle is enough to complete a certain passenger order. Thus, the state-action reward $R(s, a)$ depends on both the number of demands $\langle s, a \rangle$ and the number of agents at state s taking action a. A concise reward function is defined in Section 3.1.

The Policy and Objective. Let $\pi(a|s)$ denote the probability of taking the action $a \in A(s)$ at state s, we have $\pi(a|s) \in [0, 1]$ and $\sum_{a \in A(s)} \pi(a|s) = 1$. Let $\pi(s) = \{\pi(a|s)\}_{a \in A(s)}$ denote the local policy at the state s. A policy $\pi = \{\pi(s)\}_{s \in S}$ with $\pi(-s)$ referring to all the local policies except $\pi(s)$. For the city-scale MoD systems with a large agent population, it is not possible to compute a unique policy for each agent. Therefore, similar to previous works [27, 35], our ultimate goal is to compute a homogeneous policy π for all agents such that the total rewards over the horizon T, $\sum_{s \in S, a \in A(s)} R(s, a)$ is maximized [1]. As the number of agents in the system is large, the action probability can be interpreted as the fractional population, and converts agents into a spatio-temporal flow.

Algorithm 1: Computing the Expected Number of Agents $\lambda_\pi(s)$

Input : The policy π and target state s.
Output : The expected number of agents at s, $\lambda_\pi(s)$.

```
1 Initialize λπ(s0) = n;
2 for 1 ≤ ϕ(s') ≤ ϕ(s) do
3     λπ(s') = Σ_{s''∈pre(s'), a''∈A(s'')} λπ(s'')π(a''|s'')T(s'', a'', s').
```

3.1 The Reward Function

Before defining the reward function, we first define useful notations such as the expected number of agents at states as well as its probability distribution function.

The $\mathbb{C}$-MDP can be regarded as a directed acyclic graph (DAG), where states are nodes and transitions are directed edges. We start with sorting states S in a topological order according to the transition function. Let $pre(s) = \{s' | T(s', a, s) > 0\}$ denote the *direct* previous states of s, and $post(s) = \{s' | T(s, a, s') > 0\}$ denote the *direct* posterior states of s. Given such partial orders, the set of states S can be sorted in topological order using the depth-first search technique. Let $\phi(s) \in [0, |S|]$ denote the order priority of the state s, where $\phi(s_0) = 0$. Moreover, we define $succ(s) = \{s' | \phi(s') > \phi(s)\}$ the successor states of s, and $prio(s) = \{s' | \phi(s') < \phi(s)\}$ the prior states of s. Since the transitions directed from the state with the earlier period to the state with the later period, the earlier state has a higher order priority than the later state.

Expected Number of Agents. Given the policy π, let $\lambda_\pi(s)$ denote the expected number of agents reaching state s. If the expected number of agents $\lambda_\pi(s')$ of direct previous states $s' \in pre(s)$ is known, $\lambda_\pi(s)$ can be computed by the following recurrence formula

$$\lambda_\pi(s) = \sum_{s' \in pre(s), a' \in A(s')} \lambda_\pi(s) \pi(a'|s') T(s', a', s). \tag{1}$$

The expected number of agents $\lambda_\pi(s)$ at state s only depends on the expected number of agents and actions of these previous states $prio(s)$. Thus, $\lambda_\pi(s)$ can be updated according to the topological order, and a dynamic programming (DP)-based technique can be used to compute $\lambda_\pi(s)$, which is shown in Algorithm 1. In Line 1, all agents start from the source state s_0, i.e., the expected number of agents at s_0, $\lambda_\pi(s_0) = n$. In Lines 2-3, the expected number of agents $\lambda_\pi(s')$ of each previous state $s' \in prio(s)$ is computed in a topological order.

Similarly, let $\lambda_\pi(s, a)$ denote the expected number of the agents reaching state s taking action a, which can be computed by $\lambda_\pi(s, a) = \lambda_\pi(s) \cdot \pi(a|s)$. Intuitively, the minimum between the expected number of agents and the expected number of demands can be used for

[1] From here on in our discussion we will assume no discounting, although for completeness we do include the possibility of discounting in the algorithm.

state-action reward approximation [35], i.e.,

$$R(s,a) \approx \min\{\lambda_\pi(s,a), \tilde{o}(s,a)\}. \tag{2}$$

where $\tilde{o}(s,a) = \sum_k k \cdot o_k(s,a)$ denote the expected number of demands. Unfortunately, the linear reward function can deteriorate real reward arbitrarily on problems when there are few agents, as shown in Figure 1.

Probability Distribution of Agents. Here, using the probability distribution of agents [18], we propose a expected reward function that better approximates the real reward function. Given the expected number of agents $\lambda_\pi(s,a)$, each agent has the probability $\frac{\lambda_\pi(s,a)}{n}$ reaching state s taking action a. Thus, the probability of exactly k agents reaching state s taking action a follows a Binomial distribution:

$$f_\lambda^k(s,a) = \binom{n}{k}\left(\frac{\lambda_\pi(s,a)}{n}\right)^k\left(1-\frac{\lambda_\pi(s,a)}{n}\right)^{n-k}. \tag{3}$$

where n and $\frac{\lambda_\pi(s,a)}{n}$ represent the twin parameters, namely the number of trials and the probability of success of each trial of a Binomial distribution, respectively.

Expected Reward Function. Given the demand probability distribution $o(s,a) = \{o_0(s,a), o_1(s,a), \cdots\}$ and the agent probability distribution $f_\lambda(s,a) = \{f_\lambda^0(s,a), f_\lambda^1(s,a), \cdots\}$. the expected reward $R(s,a)$ achieved by taking action a at state s is defined by

$$\begin{aligned} R(s,a) &= \sum_{k=0}^{n} k\Big[o_k(s,a)\big(\sum_{j\geq k} f_\lambda^j(s,a)\big) + f_\lambda^k(s,a)\big(\sum_{j\geq k+1} o_j(s,a)\big)\Big] \\ &= \sum_{k=0}^{n-1}[1-F_\lambda^k(s,a)][1-O_k(s,a)] \end{aligned} \tag{4}$$

where $F_\lambda(s,a)$ and $O(s,a)$ are the Cumulative Distribution Functions of $f_\lambda(s,a)$ and $o(s,a)$, respectively. To mitigate the computation load of $R(s,a)$, we can pre-compute $F_\lambda(s,a)$ by dividing the expected number of agents $\lambda(s,a)$ into a set of intervals in an offline manner.

We use a toy example to illustrate these notations including the expected number of agents, the probability distribution of agents and the expected reward function.

Example 1. *In the left of Figure 1, there is an MoD instance consisting of six states* $\{s_0, s_1, s_2, s_3, s_4, s_5\}$. *The directed edge between states indicates the deterministic transition function. Each transition* (s, a, s') *is associated with a two-tuple* $\langle\pi(a|s), o(s,a)\rangle$, *where the former* $\pi(a|s)$ *indicates the local policy. The latter* $o(s,a) = \{o_0(s,a), o_1(s,a)\}$ *indicates the demand distribution. Here, we assume demand follows a 0-1 distribution, i.e., has a* $o_1(s,a)$ *probability, there is one demand and a* $o_0(s,a)$ *probability for zero demand. Now suppose that there are two agents starting from the source state* s_0. *Given the policy* $\pi = \{\pi(s_0), \pi(s_1), \pi(s_2), \pi(s_3), \pi(s_4), \pi(s_5)\}$, *the expected number of agents reaching state* s_2 *taking action* $\to v(s_3)$ *is* $\lambda_\pi(s_2)\cdot\pi(\to v(s_3)|s_2) = \lambda_\pi(s_0)\cdot\pi(\to v(s_2)|s_0)\cdot\pi(\to v(s_3)|s_2) = 0.4$. *The probability of zero agent reaching state* s_2 *taking action* $\to v(s_3)$ *is* $f_\lambda^0(s_2, \to v(s_3)) = \binom{2}{0}(\frac{0.4}{2})^0(1-\frac{0.4}{2})^2 = 0.64$, *the probability of one agent* $f_\lambda^1(s_2, \to v(s_3)) = 0.32$ *and the probability of two agents* $f_\lambda^0(s_2, \to v(s_3)) = 0.04$. *Finally, for the state-action* $\langle s_2, \to v(s_3)\rangle$, *the expected state-action reward*

$$R(s_2, \to v(s_3)) = \sum_{k=0}^{n-1}[1-F_k^\lambda(s,a)][1-O_k(s,a)] = 0.216.$$

Moreover, from the right of Figure 1, we can find that the expected reward function is efficient to approximate real reward function.

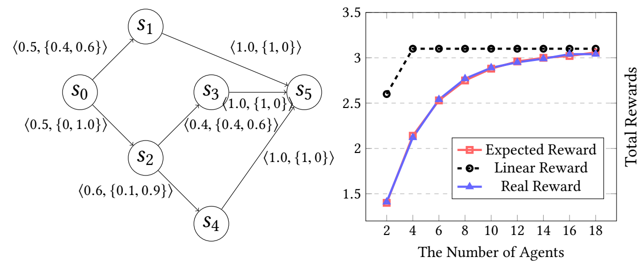

Figure 1: Left: A toy MoD instance. Right: Given the policy π, the total rewards achieved by different reward function over different number of agents. The real reward is averaged over 1000 trials, which is approximately equal to the same with the expected reward. However, there is a non-negligible error between the linear reward and real reward.

Remarks. Due to the dependence of reward on the number of agents, this optimization problem of $\mathbb{C}$-MDP becomes significantly more complicated than a single agent MDP. Using a piecewise linear reward function to approximate the expected reward function (i.e., Eq.(2)), a baseline LP solution has been proposed in [7, 18, 35]. However, the disadvantages of the LP baseline are: 1) requiring carefully designed approximation of the non-linear objective function, where the solution quality can deteriorate arbitrarily, 2) non-adaptive to real agent and demand information, and 3) time consuming of reusing the LP to return the adaptive policy at each decision period. To address these issues, this paper proposes a novel policy iteration variant, which can be adapted to dynamic MoD environments and improved for online planning in an anytime way.

4 THE ALGORITHM

In this section, by examining how the history policy affects the successive state-action reward, we propose a novel policy iteration algorithm that can find optimal solutions on the constructed $\mathbb{C}$-MDP. The proposed algorithm builds off the structured state value-based policy evaluation and the gradient-ascent-based policy improvement.

4.1 Structured State Value Function

Since the state-action reward $R(s,a)$ depends on the expected number of agents $\lambda_\pi(s)$, we first quantify how the local policy $\pi(s)$ at s affects the expected number of agents $\lambda_\pi(s')$ of the successor state $s' \in succ(s)$. From the point of view of state s, we can rewrite the policy $\pi = \langle\pi(s), \pi(-s)\rangle$. Let $\langle\pi'(s), \pi(-s)\rangle$ be a policy identical to π except to perform the local policy $\pi'(s)$ at state s. Next, we show that the local policy $\pi(s)$ has a linear effect on the expected number of agents at the successor state $s' \in succ(s)$.

Algorithm 2: Structured State Value Function $V_\pi(s)$

Input : The policy π and the target state s.
Output : The state value $V_\pi(s)$.

```
1 ∀s′ ∈ succ(s), λ_π(s′) ← Algorithm 1;
2 for s′ ∈ succ(s) do
3     V_π(s) = V_π(s) + Σ_{a′∈A(s′)} R(s′, a′);
```

Lemma 1. *Given the state s and its successor state $s' \in succ(s)$, let $\lambda_{\langle \pi(s), \pi(-s)\rangle}(s')$ denote the expected number of agents at s' under the policy $\langle \pi(s), \pi(-s)\rangle$, and $\lambda_{\langle \pi'(s), \pi(-s)\rangle}(s')$ denote the expected number of agents s' under the policy $\langle \pi'(s), \pi(-s)\rangle$. We have*

$$\begin{aligned}&\lambda_{\langle \pi(s)+\pi'(s), \pi(-s)\rangle}(s') \\ &= \lambda_{\langle \pi(s), \pi(-s)\rangle}(s') + \lambda_{\langle \pi'(s), \pi(-s)\rangle}(s'). \end{aligned} \tag{5}$$

where the policy $\langle \pi(s) + \pi'(s), \pi(-s))\rangle$ is identical to π except to perform the local policy $\pi(s) + \pi'(s)$ at state s.

This additive property will be useful for gradient computation in Section 4.2. Omitted proofs are shown in the appendix.

Structured State Value Function $V_\pi(s)$. Similar to MDPs, we require a state value function to estimate how good it is to be at a state. In $\mathbb{C}$-MDPs, considering that the state-action rewards depend on history policies, we define a new state value function variant as the total rewards accumulated from all successor state-action pairs $\langle s', a'\rangle$, i.e.,

$$V_\pi(s) = \sum_{s' \in succ(s), a' \in A(s')} R(s', a'). \tag{6}$$

The structure state value function explicitly captures the influence of history policy on related state-action rewards. In particular, the state value function at the initial state s_0, $V_\pi(s_0)$ returns the offline objective. Using the expected number of agents returned by Algorithm 1, we can evaluate the state value function $V_\pi(s)$ for an arbitrary policy π and state s. The policy evaluation is formally described in Algorithm 2. In Line 1, Algorithm 1 is used to return the expected number of agents $\lambda_\pi(s')$ at each successor state $s' \in succ(s)$. Given the expected number of agents at $succ(s')$, Lines 2-3 accumulate the total rewards of these successor state-action pairs $\langle s', a'\rangle$.

4.2 Gradient Ascent-Based Policy Improvement

We propose a gradient ascent (GA)-based policy improvement algorithm to optimize the state value function. We first introduce GA in stateless settings, and extend GA to our $\mathbb{C}$-MDPs settings. Policy gradient algorithms have widely employed in RL for action selection [31]. Different from traditional parameterized policy gradient methods, we directly calculate the policy gradient on the structured state value function without any kind of parameterization. Moreover, the proposed GA-based policy improvement algorithm is guaranteed to converge to global optima.

A straightforward extension of GA to $\mathbb{C}$-MDPs is to enumerate all of the pure strategies in $\mathbb{C}$-MDPs. Each pure strategy consists of the complete sequential deterministic action starting from the source state s_0. The policy can be a probability distribution over these pure strategies. By translating $\mathbb{C}$-MDPs into stateless settings, GA is guaranteed to converge to the optimal solution [6]. However, this kind of GA extension has an exponentially increasing strategy space with the size of states and actions [7, 15], which is impractical for city-scale MoD systems. This paper proposes a new method to scale-up as well as to guarantee convergence. The fundamental idea is to use GA to optimize the local policy $\pi(s)$ at each state s. We show that optimizing the local policy maximizes the global objective.

Algorithm 3: Compute the Gradient $\nabla V_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)$

Input : The policy π, and the target state s.
Output : The gradient of local policy $\pi(s)$.

```
1 Compute λ_⟨π(s),π(−s)⟩(s′), ∀s′ ∈ succ(s);
2 for a_i ∈ A(s) do
3     Compute λ_⟨δβ_i,π(−s)⟩(s′), ∀s′ ∈ succ(s);
4     λ_⟨π(s)+δβ_i,π(−s)⟩(s′)=λ_⟨π(s),π(−s)⟩(s′)+λ_⟨δβ_i,π(−s)⟩(s′);
5     λ_⟨π(s)−δβ_i,π(−s)⟩(s′)=λ_⟨π(s),π(−s)⟩(s′)−λ_⟨δβ_i,π(−s)⟩(s′);
6     V_⟨π^k(s)+δβ_i,π^k(−s)⟩(s) ← Algorithm 2;
7     V_⟨π^k(s)−δβ_i,π^k(−s)⟩(s) ← Algorithm 2;
8     dV_⟨π^k(s),π^k(−s)⟩(s)/da_i ← Eq.(8)
```

The proposed policy improvement algorithm executes over iterations. On each iteration k, GA is used to improve the local policy $\pi^k(s)$ with respect to the state value function $V_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)$ of state s. Let $\nabla V_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)$ denote the gradient of $V_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)$ with respect to the local policy $\pi^k(s)$, which can be computed by

$$\begin{aligned}&\nabla V_{\langle \pi^k(s), \pi^k(-s)\rangle}(s) \\ &= \frac{dV_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)}{da_1}, \cdots, \frac{dV_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)}{da_{|A(s)|}} \end{aligned} \tag{7}$$

As discussed earlier, the state value function $V_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)$ is a collective effect of joint actions, it is difficult to compute the partial gradient $\frac{dV_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)}{da_i}$ for each action a_i. Inspired by online convex optimization without a gradient [14], we use a simple approximation gradient instead:

$$\begin{aligned}&\frac{dV_{\langle \pi^k(s), \pi^k(-s)\rangle}(s)}{da_i} \\ &\approx \frac{V_{\langle \pi^k(s)+\delta\beta_i, \pi^k(-s)\rangle}(s) - V_{\langle \pi^k(s)-\delta\beta_i, \pi^k(-s)\rangle}(s)}{2\delta} \end{aligned} \tag{8}$$

where δ is a small positive real number, and β_i is a unit vector $(0, \cdots, 1, \cdots, 0)$ with the ith component is equal to 1 and 0 otherwise. The additive property of expected number of agents $\lambda_\pi(s)$ can be used to speed up the gradient computation, shown in Algorithm 3. In Lines 1 and 2, at each successive state $s' \in succ(s)$, the expected numbers of agents under the policies $\langle \pi(s), \pi(-s)\rangle$ and $\langle \delta\beta_i, \pi(-s)\rangle$ are computed respectively. In Lines 3-4, the expected number of agents at s', $\lambda_{\langle \pi(s)-\delta\beta_i, \pi(-s)\rangle}(s')$ can be computed directly by adding $\lambda_{\langle \pi(s), \pi(-s)\rangle}(s')$ and $\lambda_{\langle \delta\beta_i, \pi(-s)\rangle}(s')$.

Using the gradient direction, the local policy $\pi^k(s)$ can be improved by

$$\pi^{k+1}(s) = P(\pi^k(s) + \eta_s \nabla V_{\langle \pi^k(s), \pi^k(-s)\rangle}(s). \tag{9}$$

Algorithm 4: GA-based Policy Iteration (**GA-PI**)

Input : The $\mathbb{C}$-MDP model $\mathcal{M}$.
Output : The policy π and global objective $V_{\pi^k}(s_0)$.

1 Initialize $k = 0$ and π^k arbitrary;
2 **repeat**
3   **for** $s \in S$ **do**
4     $\eta_s = 1$;
5     $\nabla V_{\langle \pi^k(s), \pi^k(-s) \rangle}(s) \leftarrow$ Algorithm 3;
6   **repeat**
7       $\pi^{k+1}(s) = P(\pi^k(s) + \eta_s \nabla V_{\pi^k}(s))$;
8       $\eta_s = \gamma \eta_s$;
9   **until** $V_{\langle \pi^{k+1}(s), \pi^k(-s) \rangle}(s) \geq V_{\pi^k}(s)$;
10     $k = k + 1$;
11 **until** *time budget is used up*;
`// offline time budget, e.g., 2 hours`
12 Return the global objective $V_{\pi^k}(s_0) = \sum_{s,a} R(s, a)$.

where η_s is the learning rate at state s. The projection function P is utilized to project the vector $\pi^k(s) + \eta_t \nabla V_{\langle \pi^k(s), \pi^k(-s) \rangle}(s)$ to the convex domain $[0, 1]^{|A(s)|}$ where $\sum_{a \in A(s)} \pi(a|s) = 1$. Given such a positive simplex domain, a polynomial time algorithm [13] of performing Euclidean norm projection can be employed.

4.3 Anytime Policy Iteration

This section presents our GA-based policy iteration algorithm (**GA-PI**). The main idea behind GA-PI is to evaluate the current policy π^k and improve it to achieve a better policy π^{k+1} on each iteration. We also show the convergence of GA-PI.

A complete GA-PI is shown in Algorithm 4. On each iteration t, GA is used to improve the local policy $\pi^k(s)$ at state s (i.e., Lines 3-10). In Line 4, we initialize the learning rate $\eta_s = 1$ at state s. In Line 5, the policy gradient $\nabla V(\pi^t, s)$ is computed by Algorithm 3. In Lines 6-9, the learning rate η_s is carefully discounted by a discounting factor $\gamma \in [0, 1]$ such that the improved policy $\pi^{t+1}(s)$ is non-deceasing over the previous policy $\pi^k(s)$. The existence of such a learning rate is shown in [6]. The policy iteration (i.e., Lines 2-11) terminates when certain condition is satisfied, e.g., the time allotted for computing offline policy is running out.

Monotony property. On the one hand, we first show the monotonicity of the GA-PI algorithm.

Lemma 2. *GA-PI is monotonically non-deceasing on the global objective $V_\pi(s_0)$ such that $V_{\pi^{k+1}}(s_0) \geq V_{\pi^k}(s_0)$.*

Theorem 1. *The GA-PI algorithm always converges within finite iterations.*

5 REAL-TIME ONLINE PLANNING

The solutions of the GA-PI constructed on the offline $\mathbb{C} - MDP$ provide a static policy, however, cannot be dynamically adjusted to real-time agent and demand distributions. In this section, we propose an online planning algorithm to dynamically adjust the policy according to real-time observations. The main idea is that given the real-time information, the offline policy is used as a baseline plan.

Algorithm 5: Real-Time Online Planning (**Online_GA-PI**)

Input : The current $\mathbb{C}$-MDP model $\mathcal{M}^t$.
Output : The real-time online planning π_t.

1 Initialize $\pi_t(a|s) = \pi_{t-1}(a|s)$;
2 **repeat**
3   Steps 3-10 in Algorithm 4 on real $\mathbb{C}$-MDP $\mathcal{M}^t$;
4 **until** *time budget is used up*;
`// real-time budget, e.g., 5s`
5 Each agent takes the action a according to the policy $\pi_t(a|s)$ and serves the demand $\langle s, a \rangle$ if any.

We reuse GA-PI to the current plan and improve it in an anytime manner.

At the beginning of each decision period t, given the observed demands and agents' positions, we first construct an online $\mathbb{C}$-MDP $\mathcal{M}^t = \langle S^t, A^t, T^t, o^t \rangle$, shown as follows.

- The state $s \in S^t$ satisfies $t(s) \geq t$.
- Given the real traffic delay d^t_{jk} between regions v_j and v_k, the transition from the current state $s = (t, v_j)$ to the state $s' = (t + d^t_{jk}, v_k)$ is deterministic, i.e., $T(s, \rightarrow v_k, s') = 1$.
- Given the demands occurring at current state $s = (t, v)$, the demand distribution $o(s, a)$ is deterministic. For example, if there are k demands $\langle s, a \rangle$ occurring at s, $o(s, a) = \{0, 0, \cdots, 1, \cdots, 0\}$ where the kth element $o_k(s, a) = 1$.
- Given agents' positions, the initial agent distribution $\lambda_t(s)$ at current state $s = (t, v)$ is deterministic, where $\lambda_t(s)$ equals to the number of agents locating at state s.

The other model parameters (i.e., the future transition function and demand distribution) are the same with the model defined in Section 3. Given the online $\mathbb{C}$-MDPs $\mathcal{M}^t$, we employ GA-PI to restart from the baseline offline policy with the real information and improve it in the new iteration. Algorithm 5 describes how to generate the real-time online planning. In Line 2, given the real-time observations, the previous period policy π_{t-1} is used as a baseline policy π_t. In Line 2-4, the baseline policy π_t is improved by the GA-PI algorithm within time budget. The real-time budget can be set as 5 seconds, and is domain dependent. In Line 5, the agent takes the action $\langle s, a \rangle$ according to the probability $\pi_t(a|s)$. Once the agent is planned to take the action $\langle s, a \rangle$, he should be responsible for serving this type of demand.

6 EXPERIMENTS

6.1 Experiment Setup

All computations are performed on a 64-bit workstation with 64 GB RAM and a 16-core 3.5 GHz processor. All records are averaged over 40 instances, and each record is statistically significant at 95% confidence level unless otherwise specified.

6.1.1 Dataset Description. Both synthetic and real-world datasets are used in our experiments.

Synthetic Dataset (SYN). The synthetic dataset consists of 20×20 regions, 48 periods and 200 agents. The demand type $o(s, a)$

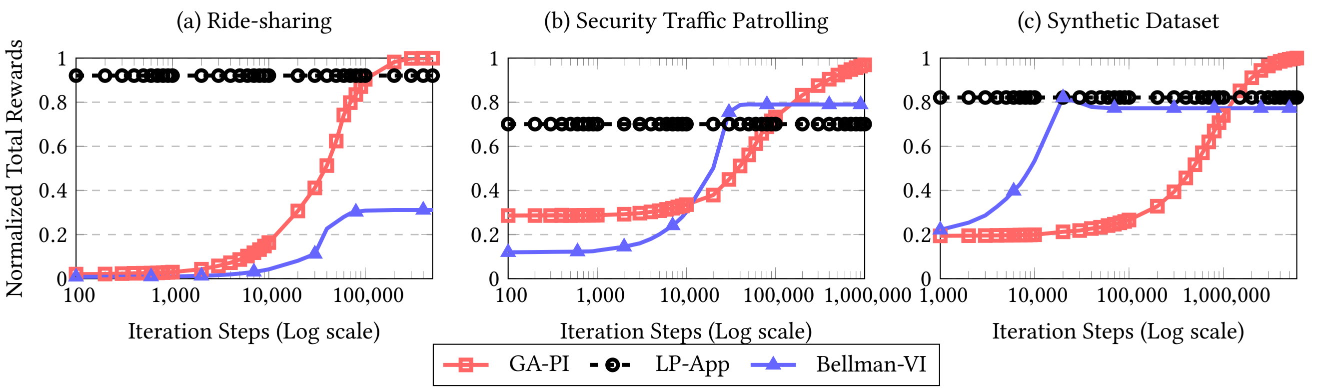

Figure 2: OFFLINE: The expected total rewards (normalized) achieved by the offline methods in real-world ride-sharing and security traffic patrolling, and the synthetic datasets.

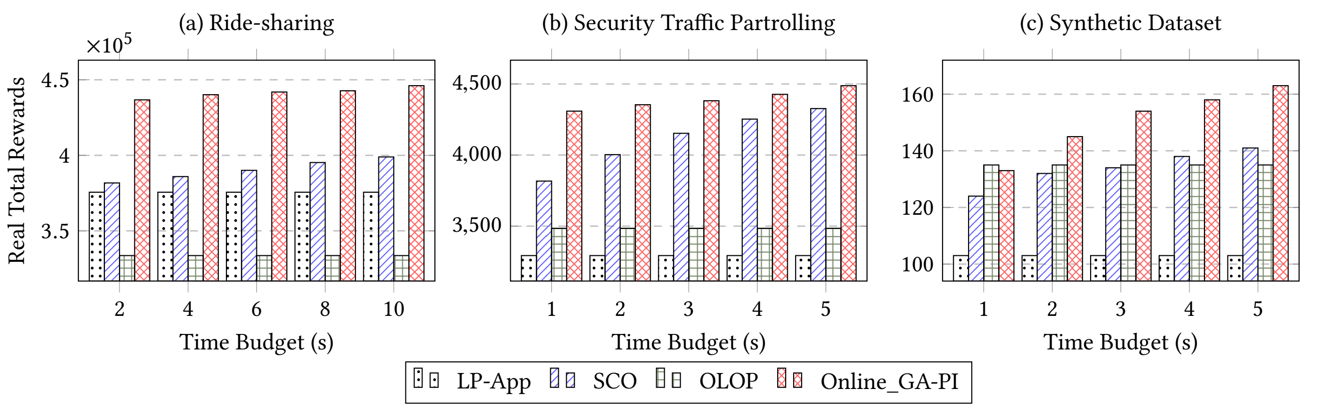

Figure 3: ONLINE: The real total rewards achieved by the online methods in real-world ride-sharing and security traffic patrolling, and the synthetic dataset.

follows a Gaussian distribution $N(1.5, 0.25)$. The transition function $T(s, a, s')$ follows the uniform distribution $U(0, 1)$ [2].

Real-World Dataset. Two typical ride-sharing and security traffic patrolling datasets are used.

• **Ride-Sharing (RS).** The ride-sharing dataset is a real trip dataset from New York city [1], which is divided into 370 non-overlapping, same-sized hexagonal regions. The horizon T is discretized into 288 periods and each period contains 5 mins. We use the trip data of February 2016, each piece of data contains the start time of a trip and the coordinates of its pick-up and drop-off regions. Pre-processing operations are conducted on the raw data, for example, some out-of-bounds trips are removed according to their coordinates of the boarding and alighting positions. After pre-processing, there are about 300,000 requests per day, which can be used to model the demand-type distribution $o(s, a)$. We use the real road network data provided by OpenStreetMap to model the transition function $T(s, a, s')$. The number of vehicles is set to 10,000.

• **Security Traffic Patrolling (STP).** The security traffic patrolling aims at the omnipresence patrolling such that when an emergency occurs, there are always police officers nearby serving it in time. The dataset is collected from a typical district of a modern city [36]. The district consists of 20×20 regions, and each day is discretized into 48 periods. The dataset contains approximately 24,000 incident requests (IRs) of the year 2017. We use the IR dataset to estimate the IR occurrence rate $o(s, a)$. The police officers taking the incident service action (i.e., staying at the current region) can respond the IR. We use OpenStreetMap to record the traffic delay, which can be used to estimate the transition function $T(s, a, s')$. There are 50 police officers available for security traffic patrolling.

6.1.2 Compared Methods. We compare our **GA-PI** method with the following two categories of methods: offline methods and online methods. Note that all methods are carefully tuned and their best results are reported.

Offline Baselines:

• LP-based approximation (**LP-App**) [35], where the linear program (LP) is constructed on $\mathbb{C} - MDP$ and the piece linear reward (i.e., Eq.(2)) is used to approximate the real reward function. The commercial solver Gurobi (version 9.10) is used to solve the LP.

• Bellman's value iteration (**Bellman-VI**) [4], where a state value function $V(s)$ is defined and improved by the Bellman formula:

$$V(s) = \max_{\pi} \sum_{a \in A_s} \pi(a|s)\big[R(s,a) + \sum_{s'} T(s,a,s')V(s')\big]. \qquad (10)$$

[2]We also test other demand and transition distributions, and have the similar results.

Online Baselines:

• LP-based approximation (**LP-App**) [12], where the online planing directly reuses the offline policy.

• Offline learning and online planning (**OLOP**) [33, 39]. Using the available historical data, a model-free Q-learning is proposed to learn the state value. Guided by the state value, an online matching between agents and demands is proposed to dispatch the agents to regions.

• Sample-Based combination optimization (**SCO**) [22], where at each period, the demands of future periods are sampled. To maximize the current and future demands of multiple samples, an integer program is proposed to return the current period policy.

6.2 Experiment Results

The Convergence in Offline Scenarios. Figure 2 shows the convergence of the proposed GA-PI to the baseline offline methods in real-world RS (Figure 2(a)) and STP (Figure 2(a)) datasets and the SYN dataset (i.e., Figure 2(a)). The x-axis represents the iteration steps, and each step records the visit and policy update at a state. The y-axis normalized total reward represents the ratio between the rewards achieved and the maximum rewards achieved by our GA-PI. From Figure 2, we can observe that in all three datasets, the proposed GA-PI method is anytime, by which the system efficiency (i.e., solution quality) increases with time. This result is in accordance with our theoretical analysis. The Bellman-VI algorithm, however, only converges to the local optima. This is in spite of Bellman-VI providing near optimal solutions in single agent MDP domains. In $\mathbb{C}$-MDPs, Bellman-VI is myopic of maximizing the current state value, ignoring the effect (i.e., the expected number of agents) of history behavior on the current state. The backward mechanism of computing the expected number of agents allows GA-PI to dominate Bellman-VI. This advantage is prominent in ride-sharing (i.e., Figure 2(a)), where customer orders (i.e., demands) are uniformly distributed over state-action pairs. By considering the history effect, the GA-PI can dispatch agents among states uniformly, but the myopic Bellman-VI always dispatches agents to states with the highest state values. In the datasets of STP and SYN with the smaller size of agents, Bellman-VI can converge faster than our proposed GA-PI. The baseline LP-App only achieves 70% rewards of GA-PI in security traffic patrolling scenario (i.e., Figure 2(b)). The potential reason is that in such a dataset, the reward $R(s, a)$ is sparse since only staying at the current regions can respond to the IR. The linear reward approximation Eq.(2) causes the mismatch between the number of agents and demands at each state, thereby decreasing the solution quality. LP-App performs the best in RS, which is followed by SYN and STP. This result can be explained by the fact that the number of agents in STP (i.e., 50) is much smaller than that in RS (i..e, 10,000). Figure 1 has verified that the linear reward function (which is used in LP-App) deteriorate the real reward significantly with few agents.

The Real-Time and Efficiency in Online Scenarios. Figure 3 compares the online methods on these datasets in terms of real-time and the exact reward achieved. The x-axis time budget represents the response time available for online methods. We observe that Online_GA-PI can always return an online plan within seconds, which can be regarded as a real-time method for MoD applications (e.g., RS and STP). Although the differences are minor, it can also be seen that given more time budget, Online_GA-PI achieves higher rewards. We argue that this can be explained by the fact that given the current period model $\mathcal{M}_t$, Online_GA-PI is an anytime method. With large enough time budget, Online_GA-PI can visit all of the state-actions properly, and it could restart from the current solution with the new information in the new iteration, and will converge to the optimal solution of $\mathcal{M}_t$. The SCO has the similar monotone property. This can be explained by the fact that given more time budget, the longer look-ahead periods can be sampled, and better solution quality will be achieved.

In most scenarios, Online_GA-PI achieves the highest rewards. An exception is in the SYN dataset with 1 second budget (i.e., Figure 3(c)), OLOP outperforms Online_GA-PI. The potential reason is that OLOP can efficiently learn the state value in such a static MoD environment (since the transition and reward function is known). As we can see in Figure 3(a) and Figure 3(b) within a more stochastic environment (the model is constructed by averaging the historical data), model-based Online_GA-PI can achieve significantly higher rewards than the model-free-based RL (i.e., OLOP). Looking at the comparisons between the offline and online scenarios, we can find the consistency property.

In summary, experiment results suggest that 1) GA-PI is an anytime algorithm that converges to the optima progressively, 2) Online_GA-PI is a real-time algorithm that scales well to city-scale MoD problems with hundreds of regions, thousands of agents and long horizon periods, and 3) Online_GA-PI outperforms state-of-the-art online planning methods in terms of solution quality.

7 CONCLUSION

This paper studies the Online_CMP problem that has a wide range of applications on MoD systems, and proposes a offline policy reuse method. In offline scenarios, the CMP is modeled as a $\mathbb{C}$-MDP where the reward is a function of the system history. Considering the effect of history behaviors on successor states, the proposed GA-PI introduces a new state value function, which can be evaluated by DP and improved by GA over iterations. We theoretically show that improving local policy increases the global objective, leading GA-PI converge to the optima. In the online stage, given the real observations, the offline policy is regarded as an initial policy and GA-PI is employed to derive an efficient online plan. Experimental results show that 1) in offline scenarios, GA-PI can converge to the optimal solution, and 2) in online scenarios, the proposed offline policy reuse can achieve efficient solution quality in real-time. As a consequence, our GA-PI technique provides a real-time Online_CMP method and theoretically guarantees that the efficiency can be improved over time.

This paper should also be viewed as providing a bridge between the offline and online collective multiagent planning. We are also encouraged by the success of offline policy reuse methods for heterogeneous multiagent systems, where the joint reward depends on coordination action of agents rather than their anonymity count.

ACKNOWLEDGMENTS

This research was supported by the National Natural Science Foundation of China (62076060, 62072099, 61932007, 61806053).

REFERENCES

[1] 2016. Taxi and limousine commission (tlc) trip record data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

[2] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 114, 3 (2017), 462–467.

[3] Richard E. Bellman. 1957. *Dynamic Programming*. Princeton University Press.

[4] Dimitri P. Bertsekas. 2005. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific.

[5] Filippo Bistaffa, Alessandro Farinelli, and Sarvapali D. Ramchurn. 2015. Sharing Rides with Friends: A Coalition Formation Algorithm for Ridesharing. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. 608–614.

[6] Stephen Boyd and Lieven Vandenberghe. 2004. Convex Optimization. Cambridge University Press, 1st edition (2004).

[7] Matthew Brown, Sandhya Saisubramanian, Pradeep Varakantham, and Milind Tambe. 2014. STREETS: Game-Theoretic Traffic Patrolling with Exploration and Exploitation. In *AAAI'14*. 2966–2971.

[8] Harshal A. Chaudhari, John W. Byers, and Evimaria Terzi. 2020. Learn to Earn: Enabling Coordination Within a Ride-Hailing Fleet. In *IEEE International Conference on Big Data, Big Data 2020, Atlanta, GA, USA, December 10-13, 2020*. 1127–1136.

[9] Shushman Choudhury, Jayesh K. Gupta, Peter Morales, and Mykel J. Kochenderfer. 2021. Scalable Anytime Planning for Multi-Agent MDPs. In *AAMAS'21*. 341–349.

[10] Daniel Claes, Frans A. Oliehoek, Hendrik Baier, and Karl Tuyls. 2017. Decentralised Online Planning for Multi-Robot Warehouse Commissioning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12*. 492–500.

[11] John P. Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. 2018. Allocation Problems in Ride-Sharing Platforms: Online Matching With Offline Reusable Resources. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI'18),New Orleans, Louisiana, USA, February 2-7, 2018*. 1007–1014.

[12] John P. Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. 2018. Assigning Tasks to Workers based on Historical Data: Online Task Assignment with Two-sided Arrivals. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*. 318–326.

[13] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. Efficient Projections onto the l1-Ball for Learning in High Dimensions. In *ICML'08*. 272–279.

[14] Abraham Flaxman, Adam Tauman Kalai, and H. Brendan McMahan. 2005. Online convex optimization in the bandit setting: gradient descent without a gradient. In *SODA'05*. 385–394.

[15] Andrew Gilpin, Samid Hoda, Javier Peña, and Tuomas Sandholm. 2007. Gradient-Based Algorithms for Finding Nash Equilibria in Extensive Form Games. In *WINE'07*, Vol. 4858. 57–69.

[16] Suining He and Kang G. Shin. 2019. Spatio-Temporal Capsule-based Reinforcement Learning for Mobility-on-Demand Network Coordination. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. 2806–2813.

[17] Jiarui Jin, Ming Zhou, Weinan Zhang, Minne Li, Zilong Guo, Zhiwei (Tony) Qin, Yan Jiao, Xiaocheng Tang, Chenxi Wang, Jun Wang, Guobin Wu, and Jieping Ye. 2019. CoRide: Joint Order Dispatching and Fleet Management for Multi-Scale Ride-Hailing Platforms. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*. 1983–1992.

[18] Rajiv Ranjan Kumar and Pradeep Varakantham. 2017. Exploiting Anonymity and Homogeneity in Factored Dec-MDPs through Precomputed Binomial Distributions. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (São Paulo, Brazil) *(AAMAS'17)*. 732–740.

[19] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. 2019. Efficient Ridesharing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. 983–994.

[20] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. 2018. Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. 1774–1783.

[21] Meghna Lowalekar, Pradeep Varakantham, Supriyo Ghosh, Sanjay Dominik Jena, and Patrick Jaillet. 2017. Online Repositioning in Bike Sharing Systems. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS'17), Pittsburgh, Pennsylvania, USA, June 18-23, 2017*. 200–208.

[22] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. 2018. Online spatio-temporal matching in stochastic and dynamic domains. *Artificial Intelligence* 261 (2018), 71 – 112.

[23] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. 2020. Competitive Ratios for Online Multi-capacity Ridesharing. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*. 771–779.

[24] Shuo Ma, Yu Zheng, and Ouri Wolfson. 2015. Real-Time City-Scale Taxi Ridesharing. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (2015), 1782–1795.

[25] Ayan Mukhopadhyay, Yevgeniy Vorobeychik, Abhishek Dubey, and Gautam Biswas. 2017. Prioritized Allocation of Emergency Responders based on a Continuous-Time Incident Prediction Model. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS'17), São Paulo, Brazil, May 8-12, 2017*. 168–177.

[26] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. 2017. Collective Multiagent Sequential Decision Making Under Uncertainty. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. 3036–3043.

[27] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. 2017. Policy Gradient With Value Function Approximation For Collective Multiagent Planning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 4319–4329.

[28] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. 2018. Credit Assignment For Collective Multiagent RL With Global Rewards. In *NeuIPS'18*. 8102–8113.

[29] Zhiwei (Tony) Qin, Xiaocheng Tang, Yan Jiao, Fan Zhang, Zhe Xu, Hongtu Zhu, and Jieping Ye. 2020. Ride-Hailing Order Dispatching at DiDi via Reinforcement Learning. *Interfaces* 50, 5 (2020), 272–286.

[30] Ariel Rosenfeld and Sarit Kraus. 2017. When Security Games Hit Traffic: Optimal Traffic Enforcement Under One Sided Uncertainty. In *IJCAI'17*, Carles Sierra (Ed.). 3814–3822.

[31] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning - an introduction*. MIT Press.

[32] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. 1057–1063.

[33] Xiaocheng Tang, Fan Zhang, Zhiwei Qin, Yansheng Wang, Dingyuan Shi, Bingchen Song, Yongxin Tong, Hongtu Zhu, and Jieping Ye. 2021. Value Function is All You Need: A Unified Learning Framework for Ride Hailing Platforms. *KDD'21* (2021).

[34] Yongxin Tong, Yuxiang Zeng, Bolin Ding, Libin Wang, and Lei Chen. 2021. Two-Sided Online Micro-Task Assignment in Spatial Crowdsourcing. *IEEE Trans. Knowl. Data Eng.* 33, 5 (2021), 2295–2309.

[35] Pradeep Varakantham, Yossiri Adulyasak, and Patrick Jaillet. 2014. Decentralized Stochastic Planning with Anonymity in Interactions. In *AAAI'14*. 2505–2512.

[36] Wanyuan Wang, Zichen Dong, Bo An, and Yichuan Jiang. 2021. Toward Efficient City-Scale Patrol Planning Using Decomposition and Grafting. *IEEE Transactions on Intelligent Transportation Systems* 22, 2 (2021), 747–757.

[37] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*. 229–256.

[38] Yuhang Xu, Wanyuan Wang, Guwei Xiong, Xiang Liu, Weiwei Wu, and Kai Liu. 2021. Network-Flow-Based Efficient Vehicle Dispatch for City-Scale Ride-Hailing Systems. *IEEE Transactions on Intelligent Transportation Systems* (2021), 1–13. https://doi.org/10.1109/TITS.2021.3054893

[39] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. 2018. Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach. In *KDD'18*. 905–913.

[40] Yisong Yue, Lavanya Marla, and Ramayya Krishnan. 2012. An Efficient Simulation-Based Approach to Ambulance Fleet Allocation and Dynamic Redeployment. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*.

[41] Ming Zhou, Jiarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, and Jieping Ye. 2019. Multi-Agent Reinforcement Learning for Order-dispatching via Order-Vehicle Distribution Matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*. 2645–2653.