Autonomous Agents and Multi-Agent Systems (2024) 38:19
https://doi.org/10.1007/s10458-024-09650-z

Offline policy reuse-guided anytime online collective multiagent planning and its application to mobility-on-demand systems

Wanyuan Wang[1] · Qian Che[1] · Yifeng Zhou[2] · Weiwei Wu[1] · Bo An[3] · Yichuan Jiang[1]

Accepted: 19 April 2024

Abstract
The popularity of mobility-on-demand (MoD) systems boosts online collective multiagent planning (Online_CMP), where spatially distributed servicing agents are planned to meet dynamically arriving demands. For city-scale MoDs with a fleet of agents, Online_CMP methods must make a tradeoff 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. Search-based online planning methods are adaptive, but are computationally expensive and cannot scale up. In this paper, we propose a principled Online_CMP 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 collective behavior of agents affects the joint reward. Given the $\mathbb{C}$-MDP model, we propose a novel state value function to evaluate the policy, and a gradient ascent (GA) technique to improve the policy. We further show that offline GA-based policy iteration (GA-PI) can converge to global optima of $\mathbb{C}$-MDP under certain conditions. Finally, with real-time information, the offline policy is used as the default plan, GA-PI is used to improve it and generate an online plan. Experimental results show that our offline policy reuse-guided Online_CMP 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 Mobility-on-demand systems · Collective multiagent planning · Policy reuse · Markov Decision process

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 [24]. In MoD systems, a fleet of agents (e.g., vehicles and police officers) are planned to meet customer demands (e.g., passengers and emergencies). Due to the uncertainty in demand arrival, online collective multiagent

Extended author information available on the last page of the article planning (Online_CMP) has attracted lots of attention [9, 10, 35]. A key characteristic of Online_CMP problems is that agents should be sequentially planned for servicing demands, and the plan in one period has a direct impact on following periods' plan.

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 directly 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, 11, 17, 36]. 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, 18, 25–28]. Since the MoD environment is non-stationary with thousands of agents learning concurrently, MARL methods are not stable and may trap into sub-optimal solutions. On the other hand, online adaptive methods proceed with a rolling horizon at each period and attempt to search a desirable plan sequence for the current observation [9, 22]. However, searching is time-consuming, online adaptive methods often have difficulty in computing policies in real-time with a sufficiently long look-ahead horizon.

In general, most existing offline learning and online search policies are independent, 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 state values. Unfortunately, in traditional OLOP framework, MARL is used to learn the state value function, which might be sub-optimal and reduce the online planning efficiency. Moreover, online one-step lookahead planning again gets trapped in the dilemma of serving current real demands or future demands (i.e., state value).

In this paper, we propose an offline police reuse-guided online Online_CMP method, which can trade computation time with solution quality. The contributions of this paper can be summarized as follows:

- We adopt the representation of a collective Markov Decision Process ($\mathbb{C}$-MDP) [25, 36] for Online_CMP in which the collective behavior of agents affects the joint reward. By examining how local policy at each state influences successive state-action reward, we design novel joint-action reward and state value functions for policy evaluation.
- We propose a gradient ascent (GA) technique to improve the offline policy monotonously. We show that the GA-based policy iteration (GA-PI) can converge to optimal solutions of $\mathbb{C}$-MDP under certain conditions. Given real-time observations, we use the offline policy as the default plan. GA-PI is then used to restart from the default plan and improve it in an anytime manner, thereby generating an efficient online plan.
- Finally, we validate our GA-PI algorithm on stochastic and deterministic MoD applications, including ride-sharing and security traffic patrolling. Experimental results show that 1) GA-PI can converge in an anytime manner, 2) The proposed Online_CMP method based on offline policy reuse significantly outperforms the state-of-the-art Online_CMP methods that separately use offline learning or online search methods. We also develop an open-source simulation platform for MoD applications.

The remainder of this paper is organized as follows. In Sect. 2, we provide a review of the related literature. In Sect. 3, we formulate the online collective multiagent planning model. We propose the offline and online planning algorithm in Sects. 4 and 5, respectively. In Sect. 6, we conduct a series of experiments to validate the proposed algorithm. Finally, we conclude our paper in Sect. 7.

2 Related work

As claimed earlier, our main contribution is to propose an offline policy reuse method for online planning, where the policy iteration technique is proposed to generate and improve the offline policy. 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, 36] and dynamic programming [37] techniques can be used to solve the offline problem. A piecewise linear approximation is presented for the non-linear reward function [17]. 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 [19], and the mean field approximation is used to model local interactions [18]. To achieve trade-off between immediate and future gains, a hierarchical RL is proposed [16], 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 [42]. The imbalance between demand and supply in future periods can also be integrated for reward design [8, 15]. By capturing the mixture of agents distribution and policy, an expectation maximization-based inference approach is proposed to optimize the policy [25]. However, these offline methods cannot be adaptive to real agent and demand distributions [10, 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 [41]. 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 [40], and Monte-Carlo Tree Search [9]. 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 [10, 11]. 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 [32]. 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 [38], policy gradient methods [26] have been widely used to learn and improve the policy parameter. Traditional parameterized policy approximation is only convergent to a locally optimal policy [31]. 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.

Ridesharing In multi-capacity ridesharing 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 [20, 23]. Multi-capacity ridesharing falls into the vehicle routing literature that makes the best greedy allocations. However, it does not consider sequential vehicle routing 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 agents $Q = \{q_1, \dots, q_n\}$ are available. The agents are planned to serve demands in sequence for T periods. In ride-sharing, agents represent vehicles and demands represent passengers [10, 11], and in security traffic patrolling, agents represent police officers and demands represent traffic/crime emergencies [24, 29]. Let $V = \{v_1, v_2, \dots, v_m\}$ be the set of m regions. Different regions have different demands and their demands vary over periods.

The response 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 [26, 36]. 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.[1] 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 $a =\rightarrow v_j$ denotes that the agent is planned to move to the region v_j.

[1] Our method can be easily extended to general settings where agents have a randomized distribution on states.

Table 1 Notation overview

Notation	Description
$Q = \{q_1, \dots, q_n\}$	The set of agents
$o(s, a)$	The probability distribution of demand
$\pi(a\|s)$	The probability of taking a at s
$\pi(s) = \{\pi(a\|s)\}_{a\in A(s)}$	The local policy at the state s
$\pi(-s)$	All the local policies except $\pi(s)$
$\pi = \langle \pi(s), \pi(-s)\rangle$	The whole system policy
$pre(s) = \{s'\|T(s', a, s) > 0\}$	The *direct* previous states of s
$post(s = \{s'\|T(s, a, s') > 0\}$	The *direct* posterior states of s
$\phi(s)$	The order priority of the state s
$succ(s) = \{s'\|\phi(s') > \phi(s)\}$	The successor states of s
$prio(s) = \{s'\|\phi(s') < \phi(s)\}$	The prior states of s
$\lambda_\pi(s)$	The expected number of agents at state s
$R_\pi(s, a)$	The expected reward of taking a at s

- *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* $o(s, a) = \{o_0(s, a), o_1(s, a), \dots\}$ 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 $a =\rightarrow v_j$ indicates the passenger order 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, 10, 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 Sect. 3.1 (Table 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 $\pi = \langle \pi(s), \pi(-s)\rangle$ denote the system policy, where $\pi(-s)$ referring to all the local policies except $\pi(s)$. For the C-MDP problem with respect to city-scale MoD systems, our object is to match the demands (i.e., vehicles) and requests (i.e., passengers) at each spatial-temporal state. This kind of object is not dependent on the identity of the agents involved, but only on the number of agents involved. Therefore, driven by the collective influence of agents, we consider the homogeneous policy π for all agents, such that the total rewards over the horizon T, $\sum_{s \in S, a \in A(s)} R(s,a)$[2] is maximized [26, 28, 36]. Because the number of agents in the MoD system is large, the action probability can be interpreted as the fractional population, and converts agents into a spatial-temporal flow.

3.1 The policy-dependent reward function

Before defining the policy-dependent 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 states with the earlier time to the states with later time, 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

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

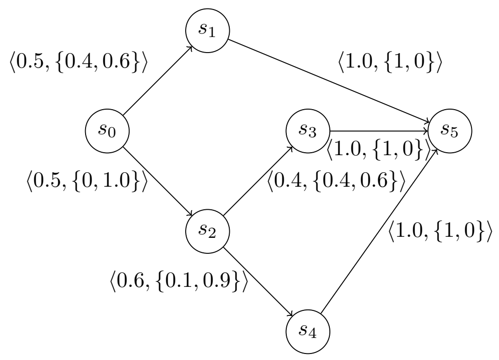
The expected number of agents $\lambda_\pi(s)$ at state s only depends on the expected number of agents and actions of these previous states $prio(s)$. Thus, $\lambda_\pi(s)$ can be updated according to the topological order, and a dynamic programming (DP)-based technique can be used to compute $\lambda_\pi(s)$, 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.

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

```
Require: The policy $\pi$ and target state $s$
Ensure: The expected number of agents at $s$, $\lambda_\pi(s)$
1: Initialize $\lambda_\pi(s_0) = n$;
2: for $1 \le \phi(s') \le \phi(s)$ do
3:     $\lambda_\pi(s') = \sum_{s'' \in pre(s'), a'' \in A(s'')} \lambda_\pi(s'')\pi(a''|s'')T(s'', a'', s')$.
4: end for
```

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

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



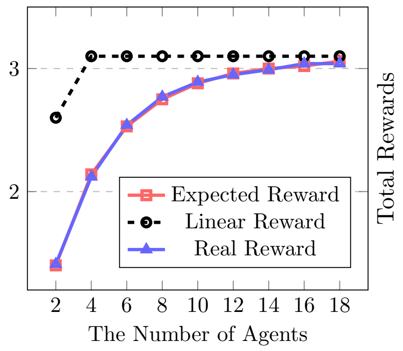


Fig. 1 *Left:* A toy MoD instance. *Right:* Given the policy π, the total rewards achieved by different reward function over different number of agents. The real reward is averaged over 1000 trials, which is approximately equal to the same with the expected reward. However, there is a non-negligible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

between the expected number of agents and the expected number of demands can be used for state-action reward approximation [36], i.e.,

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

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 with few agents, as shown in Fig. 1.

Probability distribution of agents Here, using the probability distribution of agents [17], 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 $o(s,a) = \{o_0(s,a), \dots\}$ and agent probability distribution $f_\lambda(s,a) = \{f_\lambda^0(s,a), \dots\}$ derived by the policy π, the expected reward $R(s,\ a)$ achieved by taking action a at state s can be defined by

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

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 Fig. 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 tuple $\langle \pi(a|s), o(s,a)\rangle$, where the former $\pi(a|s)$ indicates the local policy. The $o(s,a) = \{o_0(s,a), o_1(s,a)\}$ 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 $\rightarrow v(s_3)$ is $\lambda_\pi(s_2)\cdot\pi(\rightarrow v(s_3)|s_2) = \lambda_\pi(s_0)\cdot\pi(\rightarrow v(s_2)|s_0)\cdot\pi(\rightarrow v(s_3)|s_2) = 0.4$. The probability of zero agent reaching state s_2 taking action $\rightarrow v(s_3)$ is $f_\lambda^0(s_2, \rightarrow v(s_3)) = \begin{pmatrix} 2 \\ 0 \end{pmatrix}(\frac{0.4}{2})^0(1-\frac{0.4}{2})^2 = 0.64$, the probability of one agent $f_\lambda^1(s_2, \rightarrow v(s_3)) = 0.32$ and the probability of two agents $f_\lambda^0(s_2, \rightarrow v(s_3)) = 0.04$. Finally, for the state-action $\langle s_2, \rightarrow v(s_3)\rangle$, the expected state-action reward

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

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

Remark 1 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, 17, 36]. However, the disadvantages of the LP baseline are: (1) requiring carefully designed approximation of the nonlinear 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 $R_\pi(s,a)$ 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 $s' \in succ(s)$.

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

$$\lambda_{\langle \alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s') = \alpha\lambda_{\langle \pi(s),\pi(-s)\rangle}(s') + (1-\alpha)\lambda_{\langle \pi'(s),\pi(-s)\rangle}(s'). \tag{5}$$

where the policy $\langle \alpha\pi(s) + (1-\alpha)\pi'(s), \pi(-s)\rangle$ is identical to π except to perform the local policy $\alpha\pi(s) + (1-\alpha)\pi'(s)$ at state s, and $\alpha \in [0,1]$.

Proof We show it by induction with respect to states. For the state $s' \in post(s)$ of the direct posterior state of s, we have that

$$\begin{aligned}
&\lambda_{\langle \alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s') \\
&\quad = \lambda_\pi(s) \cdot (\alpha\pi(a|s) + (1-\alpha)\pi'(a|s)) \cdot T(s,a,s') \\
&\quad\quad + \textstyle\sum_{s'' \in pre(s')\backslash s, a''} \lambda_\pi(s'')\pi(a''|s'')T(s'',a'',s')
\end{aligned}$$

where $a =\rightarrow v(s')$ indicates the action taking at s and transiting to the state s', and the action $a'' =\rightarrow s'$ indicates the action taking at s'' and transiting to the state s'. Thus, we have

$$\begin{aligned}
&\lambda_{\langle \alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s') \\
&= \alpha\Big(\lambda_\pi(s)\pi(a|s)T(s,a,s') + \textstyle\sum_{s'' \in pre(s')\backslash s, a''} \lambda_\pi(s'')\pi(a''|s'')T(s'',a'',s')\Big) \\
&\quad + (1-\alpha)\Big(\lambda_\pi(s)\pi'(a|s) \cdot T(s,a,s') + \textstyle\sum_{s'' \in pre(s')\backslash s, a''} \lambda_\pi(s'')\pi(a''|s'')T(s'',a'',s')\Big) \\
&= \alpha\lambda_{\langle \pi(s),\pi(-s)\rangle}(s') + (1-\alpha)\lambda_{\langle \pi'(s),\pi(-s)\rangle}(s').
\end{aligned}$$

In the case that $s' \in succ(s)$, i.e., the state s' is the successor of s. By induction, we can assume that any prior state $s'' \in pre(s')$ of state s' has the additive property that

$$\lambda_{\langle \alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s'') = \alpha\lambda_{\langle \pi(s),\pi(-s)\rangle}(s'') + (1-\alpha)\lambda_{\langle \pi'(s),\pi(-s)\rangle}(s'').$$

For the successor state s', we have that

$$\begin{aligned}
&\lambda_{\langle \alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s') \\
&\quad= \textstyle\sum_{s''\in pre(s'),a''} \left(\lambda_{\langle \alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s'') \cdot \pi(a''|s'') \cdot T(s'',a'',s')\right) \\
&\quad= \textstyle\sum_{s''\in pre(s'),a''} \big(\alpha\lambda_{\langle \pi(s),\pi(-s)\rangle}(s'') \\
&\qquad+ (1-\alpha)\lambda_{\langle \pi'(s),\pi(-s)\rangle}(s'')\big) \cdot \pi(a''|s'') \cdot T(s'',a'',s') \\
&\quad= \alpha\lambda_{\langle \pi(s),\pi(-s)\rangle}(s') + (1-\alpha)\lambda_{\langle \pi'(s),\pi(-s)\rangle}(s').
\end{aligned}$$

Finally, we can conclude that the local policy $\pi(s)$ has the additive effect on the arrival rate $\lambda_{\langle \pi(s),\pi(-s)\rangle}(s')$ of the successor state $s' \in succ(s)$. □

This additive property will be useful for gradient computation in Sect. 4.2.

*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 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 current state-action (s, a) and all successor state-action pairs (s', a'), i.e.,

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

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 $s' \in succ(s)$. The expected number of agents $\lambda_\pi(s')$ will be used for computing immediate reward, as shown in Eq. (4). Given the expected number of agents at $succ(s')$, Lines 2–3 accumulate the total rewards of these successor state-action pairs $\langle s', a'\rangle$.

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

```
Require: The policy $\pi$ and the target state $s$
Ensure: The state value $V_\pi(s)$
1: $\forall s' \in succ(s)$, $\lambda_\pi(s') \leftarrow$ Algorithm 1;
2: $\forall a \in A(s)$, $R_\pi(s,a) \leftarrow$ Eq.(4);
3: Initialize $V_\pi(s) = \sum_{a\in A(s)} R(s,a)$;
4: for $s' \in succ(s)$ do
5:     $V_\pi(s) = V_\pi(s) + \sum_{a'\in A(s')} R_\pi(s',a')$.
6: end for
```

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 ℂ-MDPs settings. Policy gradient algorithms have widely employed in RL for action selection [32]. 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.

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, 14], 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.

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

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

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 [13], we use a simple approximation gradient instead:

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

where δ is a small positive real number, and β_i is a unit vector $(0,\dots,1,\dots,0)$ 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 $\lambda_{\langle \delta\beta_i,\pi(-s)\rangle}(s')$.

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

```
Require: The policy $\pi$, and the target state $s$.
Ensure: The gradient of local policy $\pi(s)$.
1: Compute $\lambda_{\langle \pi(s),\pi(-s)\rangle}(s')$, $\forall s' \in succ(s)$;
2: for $a_i \in A(s)$ do
3:     Compute $\lambda_{\langle \delta\beta_i,\pi(-s)\rangle}(s')$, $\forall s' \in succ(s)$;
4:     $\lambda_{\langle \pi(s)+\delta\beta_i,\pi(-s)\rangle}(s') = \lambda_{\langle \pi(s),\pi(-s)\rangle}(s') + \lambda_{\langle \delta\beta_i,\pi(-s)\rangle}(s')$;
5:     $\lambda_{\langle \pi(s)-\delta\beta_i,\pi(-s)\rangle}(s') = \lambda_{\langle \pi(s),\pi(-s)\rangle}(s') - \lambda_{\langle \delta\beta_i,\pi(-s)\rangle}(s')$;
6:     $V_{\langle \pi^k(s)+\delta\beta_i,\pi^k(-s)\rangle}(s) \leftarrow$ Algorithm 2;
7:     $V_{\langle \pi^k(s)-\delta\beta_i,\pi^k(-s)\rangle}(s) \leftarrow$ Algorithm 2;
8:     $\frac{dV_{\langle \pi^k(s),\pi^k(-s)\rangle}(s)}{da_i} \leftarrow$ Eq.(8).
9: end for
```

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

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

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 [12] of performing Euclidean norm projection can be employed.

4.3 GA-based policy iteration algorithm

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.

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.

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

```
Require: The $\mathbb{C}$-MDP model $\mathcal{M}$.
Ensure: The policy $\pi$ and global objective $V_{\pi^k}(s_0)$.
1:  Initialize $k = 0$ and $\pi^k$ arbitrary;
2:  Initialize $V_{\pi^0}(s_0) = \sum_{s\in S, a\in A(s)} R_\pi(s, a)$;
3:  while time budget is used up do
4:      for $s \in S$ do
5:          $\eta_s = 1$;
6:          $\nabla V_{\langle \pi^k(s), \pi^k(-s)\rangle}(s) \leftarrow$ Algorithm 3;
7:          while $V_{\langle \pi^{k+1}(s), \pi^k(-s)\rangle}(s) \geq V_{\pi^k}(s)$ do
8:              $\pi^{k+1}(s) = P(\pi^k(s) + \eta_s \nabla V_{\pi^k}(s))$;
9:              $\eta_s = \gamma \eta_s$;
10:         end while
11:         $k = k + 1$;
12:     end for
13: end while
14: Return the global objective $V_{\pi^k}(s_0) = \sum_{s\in S, a\in A(s)} R(s, a)$.
```

4.4 Theoretical analysis

In this section, we first show that GA-PI can converge with finite iterations, and prove its convergence to the optimum under certain conditions.

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

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

Proof Let $V_{\langle \pi^k(s_0), \pi^k(-s_0)\rangle}(s_0)$ denote the global objective on the kth iteration. Without loss of generality, assume that the source state s_0 is visited at the kth iteration, and the local policy $\pi^k(s_0)$ will be improved to $\pi^{k+1}(s_0)$ by the GA Eq. (9). Let $V_{\langle \pi^{k+1}(s_0), \pi^k(-s_0)\rangle}(s_0)$ denote the global objective on the iteration $k+1$, we have that

$$V_{\langle \pi^{k+1}(s_0), \pi^k(-s_0)\rangle}(s_0) \geq V_{\langle \pi^k(s_0), \pi^k(-s_0)\rangle}(s_0). \tag{10}$$

Next, on the $(k+1)$th iteration, for any successor state $s \in succ(s_0)$, by fixing the local policy $\pi^{k+1}(s_0)$ at source state s_0 and local policies $\pi^{k+1}(-s)$ at other states expected s, the local policy $\pi^{k+1}(s)$ at s will be improved to $\pi^{k+2}(s)$. The global objective $V_{\pi^{k+2}(s), \pi^{k+1}(-s)\rangle}(s_0)$ on the iteration $t+2$ satisfies

$$\begin{aligned} & V_{\langle \pi^{k+2}(s), \pi^{k+1}(-s)\rangle}(s_0) \\ & \quad = \sum_{s' \notin succ(s), a'} R_{\pi^{k+1}}(s', a') + V_{\langle \pi^{k+2}(s), \pi^{k+1}(-s)\rangle}(s)\langle\rangle \end{aligned} \tag{11}$$

$$\geq \sum_{s' \notin succ(s), a'} R_{\pi^{k+1}}(s', a') + V_{\langle \pi^{k+1}(s), \pi^{k+1}(-s)\rangle}(s) \tag{12}$$

$$= V_{\langle \pi^{k+1}(s_0), \pi^{k+1}(-s_0)\rangle}(s_0) \tag{13}$$

Equation (11) holds because

$$V_{\langle \pi^{k+2}(s), \pi^{k+1}(-s)\rangle}(s) = \sum_{s'' \in succ(s), a'' \in A(s')} R_{\pi^{k+2}}(s'', a'').$$

In Eq. (12) follows from the policy improvement from $\pi^{k+1}(s)$ to $\pi^{k+2}(s)$ at state s. Equation (13) holds because the update of the local policy $\pi^{k+1}(s)$ will not affect the reward of other states $s' \notin succ(s)$. Inductively, we can conclude the monotonicity result that each iteration the local policy $\pi^k(s)$ is improved to $\pi^{k+1}(s)$, the global objective will be improved as well. □

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

On one hand, according to the expected reward function $R(s, a)$ defined in Eq. (4), the global objective $V_{\pi}(s_0)$ has a upper bound $\sum_{s,a} \tilde{o}(s, a)$, where $\tilde{o}(s, a)$ is expected number of demands $\langle s, a\rangle$. Derived from the monotonicity that each iteration the policy is improved, $V_{\pi}(s_0)$ will be increased appropriately. Thus, we can conclude the convergence result.

Convergence to the optimum with linear state-action reward In the following, we show that when the state-action reward function $R(s, a)$ is approximated linearly by Eq. (2), i.e., $R_{\pi}(s, a) = \min\{\lambda_{\pi}(s, a), \tilde{o}(s, a)\}$, the GA-PI will converge to the optimum.

Given the linear reward function $R(s, a)$, We first present the concave property of the state value $V_{\pi}(s)$. By fixing the policy $\pi(-s)$ at all states except to the state s, we show that the state value function $V_{\langle \pi(s), \pi(-s)\rangle}(s)$ is concave with the local policy $\pi(s)$ at state s. Before presenting the concave property, we first present a useful proposition.

Proposition 2 *Let $f(x) = \min\{bx, c\}$, where $b \in \mathbb{R}$ and $c \in \mathbb{R}$, $f(x)$ is concave such that given any $x \neq y$ and $\alpha \in [0, 1]$: $f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha)f(y)$.*

With the help of the Proposition 2, we can prove the concave property of the state value function $V(\pi(s), \pi(-s), s)$.

Theorem 3 *Let $R_\pi(s,a) = \min\{\lambda_\pi(s,a), \tilde{o}(s,a)\}$, given the policy $\pi(-s)$ at all states except state s, the state value function $V_{\langle\pi(s),\pi(-s)\rangle}(s)$ is concave with the local policy $\pi(s)$ at state s, i.e., $\forall\alpha \in [0,1], \pi(s) \neq \pi'(s)$, we have*

$$V_{\langle\alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s) \geq \alpha V_{\langle\pi(s),\pi(-s)\rangle}(s) + (1-\alpha)V_{\langle\pi'(s),\pi(-s)\rangle}(s). \tag{14}$$

Proof We first show that the policy $\pi(s)$ is convex. For any $\pi(s) \in [0,1]^d$, $\pi'(s) \in [0,1]^d$, and $\alpha \in [0,1]$, we have $\alpha\pi(s) + (1-\alpha)\pi'(s) \in [0,1]^d$, where d is the dimension of action space at s. Given the policy $(\pi(s), \pi(-s))$, the state value function at s is

$$\begin{aligned}
&V_{\langle\pi(s),\pi(-s)\rangle}(s)\\
&\quad= \textstyle\sum_a R(s,a) + \sum_{s'\in succ(s),a'} R(s',a')\\
&\quad= \textstyle\sum_{a\in\mathcal{A}(s)} \min\{\lambda_{\langle\pi(s),\pi(-s)\rangle}(s)\pi(a|s), \tilde{o}(s,a)\}\\
&\qquad+ \textstyle\sum_{s'\in succ(s),a'} \min\{\lambda_{\langle\pi(s),\pi(-s)\rangle}(s')\pi(a'|s'), \tilde{o}(s',a')\}.
\end{aligned}$$

Alternatively, given the policy $\langle\pi'(s), \pi(-s)\rangle$, the state value function becomes

$$\begin{aligned}
&V_{\langle\pi'(s),\pi(-s)\rangle}(s)\\
&\quad= \textstyle\sum_{a\in\mathcal{A}(s)} \min\{\lambda_{\langle\pi(s),\pi(-s)\rangle}(s)\pi'(a|s), \tilde{o}(s,a)\}\\
&\qquad+ \textstyle\sum_{s'\in succ(s),a'} \min\{\lambda_{\langle\pi'(s),\pi(-s)\rangle}(s')\cdot\pi(a'|s'), \tilde{o}(s',a')\}.
\end{aligned}$$

Similarly, for the policy $\langle\alpha\pi(s) + (1-\alpha)\pi'(s), \pi(-s)\rangle$, we have

$$\begin{aligned}
&V_{\langle\alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s)\\
&\quad= \textstyle\sum_a \min\{\lambda_{\langle\pi(s),\pi(-s)\rangle}(s)\big(\alpha\pi(a|s)+(1-\alpha)\pi'(a|s)\big), \tilde{o}(s,a)\}\\
&\qquad+ \textstyle\sum_{s'\in succ(s),a'} \min\left\{\lambda_{\langle\alpha\pi(s)+(1-\alpha)\pi'(s),\pi(-s)\rangle}(s')\pi(a'|s'), \tilde{o}(s',a')\right\}\\
&\quad= \textstyle\sum_a \min\left\{\lambda_{\langle\pi(s),\pi(-s)\rangle}(s)\big(\alpha\pi(a|s)+(1-\alpha)\pi'(a|s)\big), \tilde{o}(s,a)\right\}\\
&\qquad+ \textstyle\sum_{s'\in succ(s),a'} \min\left\{\big(\alpha\lambda_{\langle\pi(s),\pi(-s)\rangle}(s')\right.
\end{aligned} \tag{15}$$

$$
\begin{aligned}
&\quad + (1-\alpha)\lambda_{\langle \pi'(s),\pi(-s)\rangle}(s')\big)\pi(a'|s'), \tilde{o}(s',a')\Big\} \\
&\geq \alpha \textstyle\sum_{a\in\mathcal{A}(s)} \min\{\lambda_{\langle \pi(s),\pi(-s)\rangle}(s)\pi(a|s), \tilde{o}(s,a)\} \\
&\quad + (1-\alpha)\textstyle\sum_{a\in\mathcal{A}(s)} \min\{\lambda_{\langle \pi(s),\pi(-s)\rangle}(s)\pi'(a|s), \tilde{o}(s,a)\} \\
&\quad + \alpha \textstyle\sum_{s'\in succ(s),a'} \min\{\lambda_{\langle \pi(s),\pi(-s)\rangle}(s')\pi(a'|s'), \tilde{o}(s',a')\} \\
&\quad + (1-\alpha)\textstyle\sum_{s'\in succ(s),a'} \min\{\lambda_{\langle \pi'(s),\pi(-s)\rangle}(s')\pi(a'|s'), \tilde{o}(s',a')\} \\
&= \alpha V(\pi(s),\pi(-s),s) + (1-\alpha)V(\pi'(s),\pi(-s),s).
\end{aligned} \tag{16}
$$

The Eq. (15) derives from additive property in the Lemma 1, and the In Eq.(16) derives from concavity of Proposition 2. Finally, we have this concave result. □

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

```
Require: The current ℂ-MDP model $\mathcal{M}^t$.
Ensure: The real-time online planning $\pi_t$.
1: Initialize $\pi_t(a|s) = \pi_{t-1}(a|s)$;
2: while time budget is used up do
3:     Steps 3-10 in Algorithm 4 on real ℂ-MDP $\mathcal{M}^t$;
4: end while
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.
```

5 Real-time online planning

The solutions of the GA-PI constructed on the offline 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. 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 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, $o(s, a) = \{0, 0, \dots, 1, \dots, 0\}$ 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 Sect. 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 sec, 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 $o(s, a)$ follows a Gaussian distribution $N(1.5, 0.25)$. The transition function $T(s, a, s')$ follows the uniform distribution $U(0, 1)$. We also test other demand and transition distributions, and have the similar results.

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 min. 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 [37]. 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

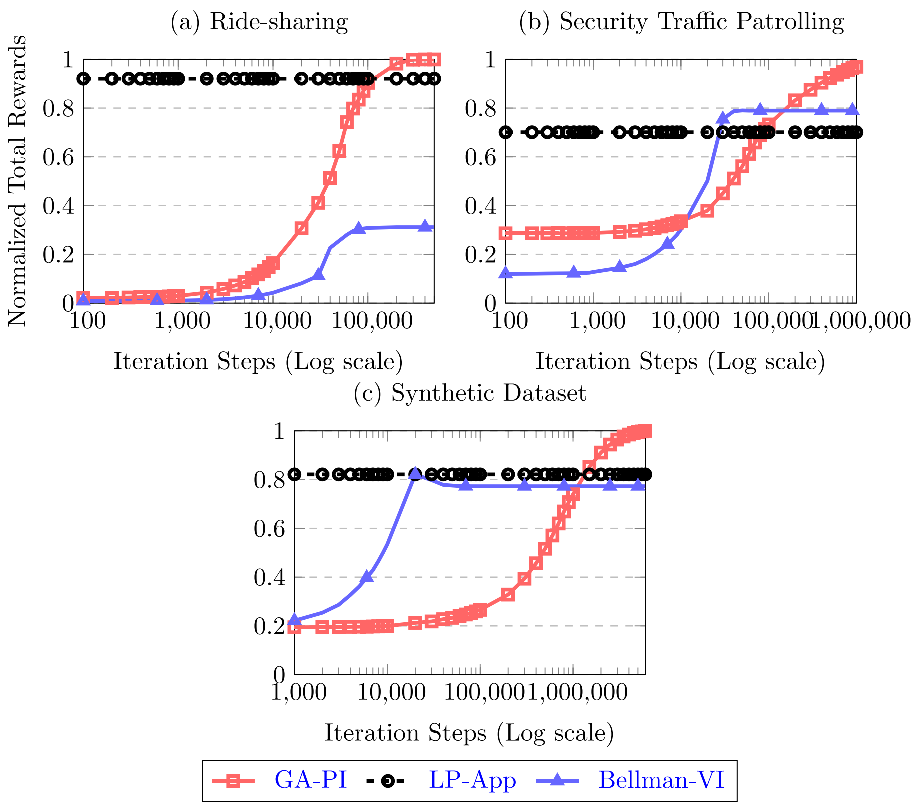


Fig. 2 *OFFLINE:* The expected total rewards (normalized) achieved by the offline methods in real-world ride-sharing and security traffic patrolling, and the synthetic datasets

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 Evaluation metric

We use the accumulated (expected) reward and computation time to evaluate the performance. The reward represents the passenger orders served in ride-sharing and incidents responded in security traffic patrolling. In terms expectation, we mean in the offline scenarios.

6.1.3 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*) [36], 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:

$$V(s) = \max_{\pi} \sum_{a \in A_s} \pi(a|s) \left[R(s,a) + \sum_{s'} T(s,a,s')V(s') \right]. \tag{17}$$

Online baselines:

- LP-based approximation (*LP-App*) [11], 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 Fig. 2 shows the convergence of the proposed GA-PI to the baseline offline methods in real-world RS (Fig. 2a) and STP (Fig. 2a) datasets and the SYN dataset (i.e., Fig. 2a). 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 Fig. 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., Fig. 2a), 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., Fig. 2b). 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

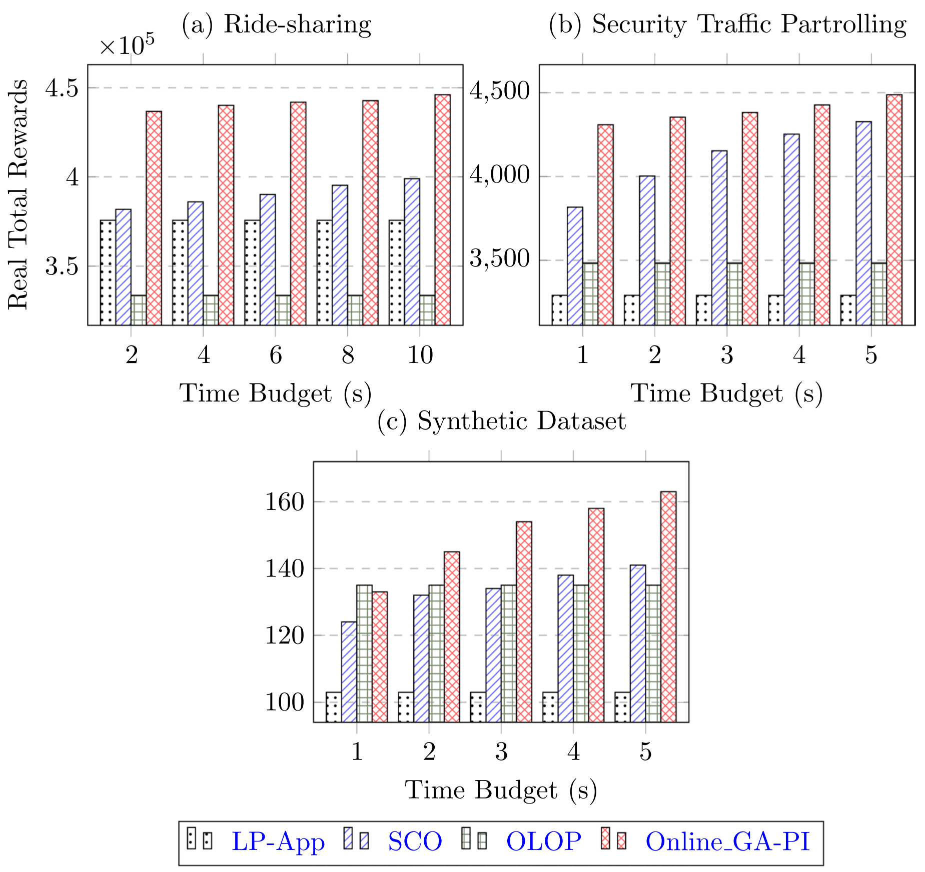


Fig. 3 *ONLINE:* The real total rewards achieved by the online methods in real-world ride-sharing and security traffic patrolling, and the synthetic dataset

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 Fig. 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 s budget (i.e., Fig. 3c), 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 Fig. 3a, 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.

6.3 Comparisons on deterministic domain

6.3.1 Problem description

Traffic enforcement problem (TEP) A set of police officers is deployed for traffic enforcement, with the aim of preventing road accidents or drivers' illegal behaviors [30]. The interaction between police officers and drivers as a defender-attacker Stackelberg game. The defender (the police) commits to a deterministic pure patrol strategy on a road network $G = \langle V, E \langle$, where $V = \{v\}$ is the set of intersections and $E = \{e = (u, v)\}$ is the set of road segments. A daily patrol schedule consists of a trajectory through the graph, i.e., a sequence of road segments to patrol. The attacker (drivers, thereby accidents) follows the opportunistic crimes model, and reactive to police enforcement in the current and past periods.

Following the recent advancement in predictive policing, the risk of accidents occurring at the road-segment and period t (e^t for short), as $\text{risk}(e^t) \in [0, 1]$. This risk function measures how likely a serious traffic accident likely to occur at e^t in the absence of police enforcement. At each period t, the police places enforcement on a subset of size n from E (n denote the number of police officers). We denote the traffic police enforcement history at period t as H_t. We use the notation $H[e^t]$ as an indicator of whether a police officer is assigned to e^t. The effectiveness of enforcement as $\text{eff}(H_t, e^t) \in [0, 1]$, which measures that the effect the police enforcement history has on the risk of accidents occurring at e^t.

The traffic enforcement is interested in minimizing the total expected number of accidents (i.e., risk) occurring throughout the game. Formally, it seeks to minimize the following objective:

$$\min_{H_T} \sum_{1 \le t \le T} \sum_{e \in E} \text{risk}(e^t)(1 - \text{eff}(H_t, e^t)). \tag{18}$$

6.3.2 Compared methods

We first extend the Online_GA-PI to TEP, shown as the following steps.

Table 2 The effect of the game round T with respect to (expected) accidents reduction (AR) and running time (RT)

Methods	$T = 30$		$T = 50$		$T = 70$	
	AR	RT(s)	AR	RT(s)	AR	RT(s)
Naive	42.3	90	69.6	427	98.4	1138
ROSE	42.1	395	69.6	1510	N/A	N/A
Random	30	0	50.1	0	68.3	0
Online_GA-PI	40.1	2.7	67.1	3	95.6	2.8

Each cell is averaged over 20 instances and statistically significant at 95% confidence level. N/A indicates the computation time exceeds 1 hr

- Step 1: For each policy officer, we randomly assign the deterministic patrolling policy at each spatial-temporal state. This randomized policy can be regarded as the offline policy.
- Step 2: Given the offline policy, at each period, starting from the initial state, each police officer alternatives his policy if it can reduce the accidents [defined in (18)].
- Step 3: Executing Step 2 iteratively, until the real-time condition is satisfied (e.g., the time budget is used up).

Online_GA-PI is compared with 3 baseline methods: first, a *Naive* models the TEP as a Binary Integer Programming (BIP), which is solved by the commercial solver Gurobi. Second, a *Random* solver which for each police unit selects an action at random at each time step. Third, a *ROSE* solver [30], which consists of Master/Slave two levels: at the Master level, a relaxed BIP is solved in which only a subset of terms are introduced, and the Slave level checks whether any term should be triggered. It should be noted that Naive and ROSE methods are exact optimal solutions.

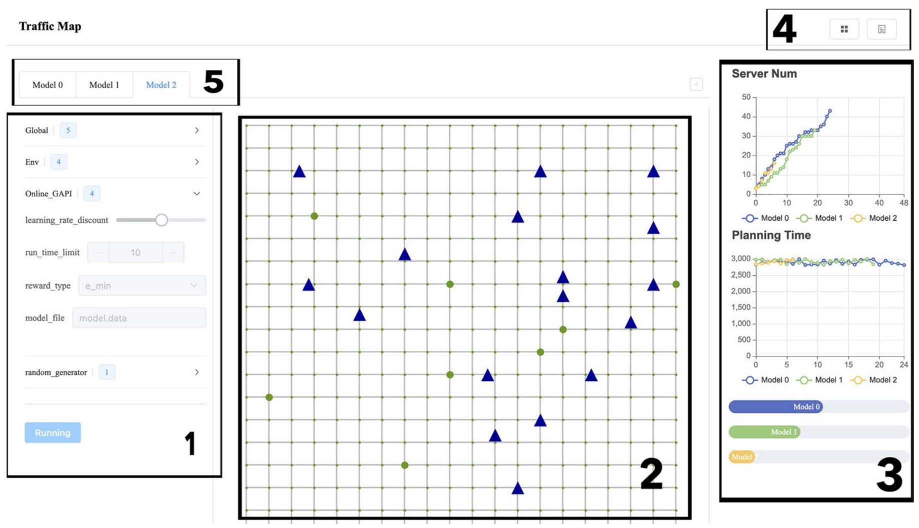
6.3.3 Evaluation metric

We use the solution quality and computation time to evaluate the performance of these methods. The solution quality is defined by the accidents reduction compared to the scenarios in the absence of police enforcement. The larger solution quality, the better the method performs.

6.3.4 Results

Table 2 shows the results on a road network with 284 intersections, 355 road-segments, and 10 police officers. The risk(e^t) $\sim N(0.15, 0.05)$ follows a norm distribution. From Table 2, we can find that (1) Naive and ROSE can reduce the most accidents, which are followed by our Online_GA-PI. (2) For the fine granularity periods (i.e., $T = 70$), Naive will consume nearly 20 min to return the final solution, and ROSE even cannot scale up, on the other hand our Online_GA-PI can return the solution at each period in a real-time manner (i.e., $\sim$ 3 s). (3) Although Random can return the real-time patrolling policy, it performs arbitrarily poorly on reducing accidents.

Concluding, our Online_GA-PI is a good option for TEP problem with respect to the tradeoff between accident reduction and computation time.



Global 5
Env 4
Online_GAPI 4
learning_rate_discount
run_time_limit 10
reward_type e_min
model_file model.data
random_generator 1

Fig. 4 Front of system: in order, the areas are parameters configuration, animation effect, indicators of system, extra function, simulation switch

6.4 A prototype system for MoDs

This section we will introduce a simulation system[3] for the online MoDs. This system mainly provides the following features for user.

1. The user can archive the designed algorithm in the standard environment.
2. The user can run the simulation with any available algorithm.
3. The user can run multiple simulations at the same time.
4. The user can observe the result and the process of the simulations, and compare with others.

With these features and the MoDs that previously described, we design an interactive system based on web. Based on the described system design and implementation, this part will show the function of the simulation system.

The frontend of system is shown in Fig. 4. In the 1st area, the users can configure the parameters of environment, algorithm and generator. The "Run" button should be clicked to notify the backend to create and start the simulation task. The 2nd area is to show the results when the simulation process is running, and which Fig. 4 shows is "Grid". The triangular objects represent agents, while circular objects are regions whose different sizes mean the number of rewards in the region. The edges between regions are roads that agent can travel through. The 3rd area shows the results of the simulation, and below the charts they are the progress bars for all running simulations. There are two button to open different function in the 4th area. The left one is to open the configure page of the animation and road network, while another one can open the simulation history list. The last one is used

[3] https://github.com/TrafficRun.

to switch the running simulations, and with the rightmost button user can add a simulation instance.

7 Conclusion and future work

This paper studies the Online_CMP problem that has a wide range of applications on MoD systems, and proposes an offline policy reuse method. In offline scenarios, the CMP is modeled as a $\mathbb{C}$-MDP where the reward policy-dependent. 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 on stochastic and deterministic domains show that (1) in offline scenarios, GA-PI can converge to the optimal solution, and (2) in online scenarios, the proposed offline policy reuse method 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. Directions for future research include offline policy reuse methods for general multiagent systems with heterogeneous agents, where the joint reward depends on coordination action of agents rather than their anonymity count. Such generalized police reuse will arise issues related to the quality of the source policy or even negative transfer, which have been studied in transfer learning [34].

Acknowledgements This research was supported in part by the Key Research and Development Projects in Jiangsu Province under Grant BE2021001-2, and in part by the National Natural Science Foundation of China (62076060, 62072099, 61932007, 61806053).

Author Contributions Wanyuan Wang and Qian Che designed the algorithm and implemented the experiments, Yifeng Zhou, Bo An, and Yichuan Jiang wrote the main manuscript text, and Weiwei Wu analyzed the algorithm.

Funding This research was supported in part by the Key Research and Development Projects in Jiangsu Province under Grant BE2021001-2, and in part by the National Natural Science Foundation of China (62076060, 62072099, 61932007, 61806053).

Availability of data and materials The dataset used will be open-sourced once this paper is accepted.

Declarations

Conflict of interest The authors declared that they have no Conflict of interest to this work.

Ethical approval Not applicable.

References

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

2. Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences, 114*(3), 462–467.
3. Bellman, R. E. (1957). *Dynamic programming, adaptive computation and machine learning*. Princeton University Press.
4. Bertsekas, D. P. (2005). *Dynamic programming and optimal control* (3rd ed.). Athena Scientific.
5. Bistaffa, F., Farinelli, A. & Ramchurn, S. D. (2015), Sharing rides with friends: A coalition formation algorithm for ridesharing. In *Proceedings of the 29th AAAI conference on artificial intelligence, January 25–30, 2015, Austin, TX, USA* (pp. 608–614).
6. Boyd, S., & Vandenberghe, L. (2004). *Convex optimization* (1st Ed.). Cambridge University Press.
7. Brown, M., Saisubramanian, S., Varakantham, P., & Tambe M. (2014). STREETS: Game-theoretic traffic patrolling with exploration and exploitation. In *AAAI'14* (pp. 2966–2971).
8. Chaudhari, H. A., Byers, J. W. & Terzi, E. (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* (pp. 1127–1136).
9. Claes, D., Oliehoek, FA., Baier, H., & Tuyls, K. (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* (pp. 492–500).
10. Dickerson, J. P., Sankararaman, K. A., Srinivasan, A., & Xu, P. (2018a). Allocation problems in ride-sharing platforms: Online matching with offline reusable resources. In *Proceedings of the 32nd AAAI conference on artificial intelligence (AAAI'18), New Orleans, Louisiana, USA, February 2–7, 2018* (pp. 1007–1014).
11. Dickerson, J. P., Sankararaman, K. A., Srinivasan, A., & Xu, P. (2018b). 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* (pp. 318–326).
12. Duchi, J., Shalev-Shwartz, S., Singer, Y., & Chandra, T. (2008). Efficient projections onto the l1-ball for learning in high dimensions. In *ICML'08* (pp. 272–279).
13. Flaxman, A., Kalai, A. T. & McMahan, H. B. (2005). Online convex optimization in the bandit setting: gradient descent without a gradient. In *SODA'05* (pp. 385–394).
14. Gilpin, A., Hoda, S., Peña, J., & Sandholm, T. (2007). Gradient-based algorithms for finding Nash equilibria in extensive form games. In *WINE'07* (pp. 57–69).
15. He, S. & Shin, K. G. (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* (pp. 2806–2813).
16. Jin, J., Zhou, M., Zhang, W., Li, M., Guo, Z., Qin, Z., Jiao, Y., Tang, X., Wang, C., Wang, J., & Wu, G. (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* (pp. 1983–1992).
17. Kumar, R. R. & Varakantham, P. (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, AAMAS'17* (pp. 732–740).
18. Li, M., Qin, Z., Jiao, Y., Yang, Y., Wang, J., Wang, C., Wu, G., & Ye, J. (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* (pp. 983–994).
19. Lin, K., Zhao, R., Xu, Z., & Zhou, J. (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* (pp. 1774–1783).
20. Lowalekar, M., Varakantham, P. & Jaillet, P. (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* (pp. 771–779).
21. Lowalekar, M., Varakantham, P., Ghosh, S., & Jena, S., & Jaillet, P. (2017). Online repositioning in bike sharing systems. In *Proceedings of the 27th international conference on automated planning and scheduling (ICAPS'17), Pittsburgh, Pennsylvania, USA, June 18–23, 2017* (pp. 200–208).
22. Lowalekar, M., Varakantham, P., & Jaillet, P. (2018). Online spatio-temporal matching in stochastic and dynamic domains. *Artificial Intelligence, 261*, 71–112.
23. Ma, S., Zheng, Y., & Wolfson, O. (2015). Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering, 27*(7), 1782–1795.

24. Mukhopadhyay, A., Vorobeychik, Y., & Dubey, A. (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* (pp. 168–177).
25. Nguyen, D. T., Kumar, A. & Lau, H. C. (2017a). Collective multiagent sequential decision making under uncertainty. In *Proceedings of the 31st AAAI conference on artificial intelligence, February 4–9, 2017, San Francisco, California, USA* (pp. 3036–3043).
26. Nguyen, D. T., Kumar, A. & Lau, H. C. (2017b). 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* (pp. 4319–4329).
27. Nguyen, D. T., Kumar, A., & Lau, H. C. (2018). Credit assignment for collective multiagent rl with global rewards. In *NeuIPS'18* (pp. 8102–8113).
28. Qin, Z. T., Tang, X., Jiao, Y., Zhang, F., Xu, Z., Zhu, H., & Ye, J. (2020). Ride-hailing order dispatching at didi via reinforcement learning. *Interfaces, 50*(5), 272–286.
29. Rosenfeld, A. & Kraus, S. (2017). When security games hit traffic: Optimal traffic enforcement under one sided uncertainty. In Sierra C (Ed.), *IJCAI'17* (pp. 3814–3822).
30. Rosenfeld, A., Maksimov, O., & Kraus, S. (2020). When security games hit traffic: A deployed optimal traffic enforcement system. *Artificial Intelligence, 289*(103), 381.
31. Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (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]* (pp. 1057–1063).
32. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning—An introduction. Adaptive computation and machine learning*. MIT Press.
33. Tang, X., Zhang, F., Qin, Z., Wang, Y., Shi, D., Song, B., Tong, Y., Zhu, H., & Ye, J. (2021). Value function is all you need: A unified learning framework for ride hailing platforms. In *KDD'21*.
34. Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research, 10*, 1633–1685.
35. Tong, Y., Zeng, Y., Ding, B., Wang, L., & Chen, L. (2021). Two-sided online micro-task assignment in spatial crowdsourcing. *IIEEE Transactions on Knowledge and Data Engineering, 33*(5), 2295–2309.
36. Varakantham, P., Adulyasak, Y. & Jaillet, P. (2014). Decentralized stochastic planning with anonymity in interactions. In *AAAI'14* (pp. 2505–2512).
37. Wang, W., Dong, Z., An, B., & Jiang, Y. (2021). Toward efficient city-scale patrol planning using decomposition and grafting. *IEEE Transactions on Intelligent Transportation Systems, 22*(2), 747–757.
38. Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine learning* (pp. 229–256).
39. Xu, Z., Li, Z., Guan, Q., Zhang, D., Li, Q., Nan, J., Liu, C., Bian, W., & Ye, J. (2018). Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *KDD'18* (pp. 905–913).
40. Xu, Y., Wang, W., Xiong, G., Liu, X., Wu, W., & Liu, K. (2021). Network-flow-based efficient vehicle dispatch for city-scale ride-hailing systems. *Transactions on Intelligent Transportation Systems*. https://doi.org/10.1109/TITS.2021.3054893
41. Yue, Y., Marla, L. & Krishnan, R. (2012). An efficient simulation-based approach to ambulance fleet allocation and dynamic redeployment. In *Proceedings of the 26th AAAI conference on artificial intelligence, July 22–26, 2012, Toronto, ON, Canada*.
42. Zhou, M., Jin, J., Zhang, W., Qin, Z., Jiao, Y., Wang, C., Wu, G., Yu, Y., & Ye, J. (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* (pp. 2645–2653).

Authors and Affiliations

Wanyuan Wang[1] · Qian Che[1] · Yifeng Zhou[2] · Weiwei Wu[1] · Bo An[3] · Yichuan Jiang[1]

✉ Qian Che
qche@seu.edu.cn

Wanyuan Wang
wywang@seu.edu.cn

Yifeng Zhou
yfzhou@nau.edu.cn

Weiwei Wu
weiweiwu@seu.edu.cn

Bo An
boan@ntu.edu.sg

Yichuan Jiang
yjiang@seu.edu.cn

[1] School of Computer Science and Engineering, Southeast University, Nanjing 211189, China

[2] School of Computer Science, Nanjing Audit University, Nanjing 211815, China

[3] School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798, Singapore