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 (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:

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.

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 methods are not stable and may trap into suboptimal 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 this paper, we trade computation time with solution quality by proposing an anytime policy iteration (PI) algorithm. We adopt the representation for a collective Markov Decision Process (C-MDP)
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 \( \pi \) [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 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 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/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 (C-MDP) representation [27, 35]. Different from the single agent MDP, the reward function in C-MDP depends on history policy rather than on merely current state and action. Formally, a C-MDP can be described by \( M = (S, A, T, R, \pi, s_0) \):

- **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 \( a \rightarrow v_j \) 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.** \( o(s, a) = \{o_0(s, a), o_1(s, a), \ldots\} \) denotes the probability distribution of demand \( (s, a) \), where \( o_k(s, a) \in [0, 1] \) is the probability of \( k \) demands \( (s, a) \) requested at \( s \) and \( \sum_k o_k(s, a) = 1 \). The demand \( o(s, a) \) can be served by the agent taking action \( a \) at state \( s \). For example, in ride-sharing, the demand \( (s, a) \), where \( a \rightarrow v_j \) 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 \( (s, a) \) 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 \( \pi \) for all agents such that the total rewards over the horizon \( T \), \( \sum_{s \in S} R(s, a) \) 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.

### 3.1 The Reward Function

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

The 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 \( pre(s) = \{ s' \mid T(s', a, s) > 0 \} \) denote the direct previous states of \( s \), and \( post(s) = \{ s' \mid T(s, a, s') > 0 \} \) 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 \( succ(s) = \{ s' \mid \phi(s') > \phi(s) \} \) the successor states of \( s \), and \( prio(s) = \{ s' \mid \phi(s') < \phi(s) \} \) 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 \( \pi \), 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.,

\[ R(s, a) \approx \min \{ \lambda \pi \cdot \bar{o}(s, a), \bar{o}(s, a) \}. \quad (2) \]

where \( \bar{o}(s, a) = \sum_k k \cdot o_k(s, a) \) 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 new reward function that better approximates the real reward function. Given the expected number of agents \( \lambda \pi \cdot \bar{o}(s, a) \), each agent has the probability \( \frac{\lambda \pi \cdot \bar{o}(s, a)}{n} \) reaching state \( s \) taking action \( a \). Thus, the probability of exactly \( k \) agents reaching state \( s \) taking action \( a \) follows a Binomial distribution:

\[ f_k^n(s, a) = \binom{n}{k} \frac{\lambda \pi \cdot \bar{o}(s, a)}{n}^k (1 - \frac{\lambda \pi \cdot \bar{o}(s, a)}{n})^{n-k}. \quad (3) \]

where \( n \) and \( \frac{\lambda \pi \cdot \bar{o}(s, a)}{n} \) 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 \( o(s, a) = \{ o_0(s, a), o_1(s, a), \ldots \} \) and the agent probability distribution \( f_j^n(s, a) = \{ f_0^n(s, a), f_1^n(s, a), \ldots \} \), the expected reward \( R(s, a) \) achieved by taking action \( a \) at state \( s \) is defined by

\[ R(s, a) = \sum_{k=0}^{n} k \left[ \sum_{j \geq k} f_j^n(s, a) \right] o_j(s, a) + \sum_{j > k} f_j^n(s, a) \left( \sum_{j \geq k+1} o_j(s, a) \right) \]

where \( F_k(s, a) \) and \( O(s, a) \) are the Cumulative Distribution Functions of \( f_k^n(s, a) \) and \( o(s, a) \), respectively. To mitigate the computation load of \( R(s, a) \), we can pre-compute \( F_k(s, a) \) by dividing the expected number of agents \( \lambda \pi \cdot o(s, a) \) 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 \( (\pi(a), o(s, a)) \), where the former \( \pi(a) \) indicates the local policy. The latter \( o(s, a) \) is \( \{ o_0(s, a), o_1(s, a), \ldots \} \) indicates the demand distribution. Here, we assume demand follows a distribution:

![Figure 1: A toy MoD instance.](image)

The Number of Agents

![Figure 1: A toy MoD instance.](image)

The expected state-action reward

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

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

**Remarks.** Due to the dependence of reward on the number of agents, this optimization problem of 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 C-MDP. The proposed algorithm builds on 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 \( \lambda \pi \cdot o(s) \), we first quantify how the local policy \( \pi(s) \) at state \( s \) affects the expected number of agents \( \lambda \pi' \cdot o(s') \) of the successor state \( s' \in \text{succ}(s) \). From the point of view of state \( s \), we can rewrite the policy \( \pi = (\pi(s), \pi(\rightarrow s)) \). Let \( (\pi'(s), \pi(\rightarrow s)) \) be a policy identical to \( \pi \) 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 \( s' \in \text{succ}(s) \).
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 [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 C-MDPs is to enumerate all of the pure strategies in 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 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.

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 $V(\pi^k(s), \pi^k(s))$ of state $s$. Let $\nabla V(\pi^k(s), \pi^k(s))$ denote the gradient of $V(\pi^k(s), \pi^k(s))$ with respect to the local policy $\pi^k(s)$, which can be computed by

$$\nabla V(\pi^k(s), \pi^k(s)) = \frac{dV(\pi^k(s), \pi^k(s))}{da_1}, \ldots, \frac{dV(\pi^k(s), \pi^k(s))}{da_{|A(s)|}} \quad (7)$$

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

$$\frac{dV(\pi^k(s), \pi^k(s))}{da_i} \approx \frac{V(\pi^k(s) + \delta \beta_i, \pi^k(s)) - V(\pi^k(s) - \delta \beta_i, \pi^k(s))}{2\delta} \quad (8)$$

where $\delta$ is a small positive real number, and $\beta_i$ is a unit vector $(0, \ldots, 1, \ldots, 0)$ with the $i$th component is equal to 1 and 0 otherwise. The additive property of expected number of agents $\lambda(s)$ can be used to speed up the gradient computation, shown in Algorithm 3. In Lines 1 and 2, at each successive state $s' \in \text{succ}(s)$, 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 $s'$, $\lambda(\pi(s) - \delta \beta_i, \pi(-s))$ is computed directly by adding $\lambda(\pi(s) - \delta \beta_i, \pi(-s))(s')$ and $\lambda(\delta \beta_i, \pi(-s))(s')$.

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

$$\pi^{k+1}(s) = P(\pi^k(s) + \eta \nabla V(\pi^k(s), \pi^k(s))) \quad (9)$$

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

**Input:** The policy $\pi$ and the target state $s$.

**Output:** The state value $V_\pi(s)$.

1. For $s' \in \text{succ}(s)$, $\lambda_\pi(s') \leftarrow$ Algorithm 1;
2. For $s' \in \text{succ}(s)$ do
3. $$V_\pi(s) = V_\pi(s) + \sum_{a' \in A(s')} R(s', a');$$

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

$$\lambda_\pi(s, \pi(-s))(s') = \lambda_\pi(s')(s') + \lambda_\pi(s, \pi(-s))(s'), \quad (5)$$

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

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

Structural 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 all successor state-action pairs $(s', a')$, i.e.,

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

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 $\pi$ and state $s$. The policy evaluation is formally discussed in Algorithm 2. In Line 1, Algorithm 1 is used to return the expected number of agents $\lambda_\pi(s')$ at each successor state $s' \in \text{succ}(s)$. Given the expected number of agents at $\text{succ}(s')$, Lines 2-3 accumulate the total rewards of these successor state-action pairs $(s', a')$. 

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 [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 C-MDPs is to enumerate all of the pure strategies in 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 C-MDPs into stateless settings,
Return the global objective \( \eta \). 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.

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

**Input**: The C-MDP model \( M \).

**Output**: The policy \( \pi \) and global objective \( V_{\pi,k}(s_0) \).

1. Initialize \( k = 0 \) and \( \pi_k \) arbitrary;
2. repeat
   3. for \( s \in S \) do
      4. \( \eta_s = 1 \);
      5. \( \nabla V(\pi_k(s), \pi_k(-s)))(s) \leftarrow \text{Algorithm 3}; \)
   6. repeat
      7. \( \pi^{k+1}(s) = P(\pi_k(s) + \eta_s \nabla V_{\pi_k}(s)); \)
      8. \( \eta_s = \alpha \eta_s; \)
      9. until \( V(\pi^{k+1}(s), \pi_k(-s)))(s) \geq V_{\pi_k}(s); \)
   10. \( k = k + 1; \)
3. until time budget is used up;
   // offline time budget, e.g., 2 hours
4. Return the global objective \( V_{\pi_k}(s_0) = \sum s, a R(s, a) \).

where \( \eta_s \) is the learning rate at state \( s \). The projection function \( P \) is utilized to project the vector \( \pi^k(s) + \eta_s \nabla V(\pi_k(s), \pi_k(-s))(s) \) to the convex domain \([0, 1]|A(s)|\) such that \( \sum a \in A(s) \pi(a(s)) = 1 \). Given such a convex 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 \( \pi_k \) and improve it to achieve a better policy \( \pi^{k+1} \) 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 \( \nabla V(\pi^k(s), \pi(-s))(s) \) is computed by Algorithm 3. In Lines 6-9, the learning rate \( \eta_s \) is carefully discounted by a discounting factor \( \gamma < 0 \) such that the improved policy \( \pi^{k+1}(s) \) is non-decreasing 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-decreasing 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 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 \( M^t = (S', A', T', o', \xi) \), shown as follows.

- The state \( s \in S' \) satisfies \( l(s) \leq t \).
- The real traffic delay \( d^T_{jk} \) between regions \( v_j \) and \( v_k \), the transition from the current state \( s = (t, u_j) \) 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 \( (s, a) \) occurring at \( s \), \( o(s, a) = \{ 0, 0, \cdots, 1, \cdots, 0 \} \) where the \( k \)th element \( o_k(s, a) = 1 \).
- Given agents’ positions, the initial agent distribution \( \lambda_i(s) \) at current state \( s = (t, v) \) is deterministic, where \( \lambda_i(s) \) 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 C-MDPS \( M^t \), 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 \( \pi_{t-1} \) is used as a baseline policy \( \pi_t \). In Line 2-4, the baseline policy \( \pi_t \) 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 \( (s, a) \) according to the probability \( \pi_t(a|s) \). Once the agent is planned to take the action \( (s, a) \), 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 \times 20 regions, 48 periods and 200 agents. The demand type \( o(s, a) \)
The security traffic patrolling aims at the omnipresence patrolling such that when an emergency occurs, there are always police officers nearby serving. The transition function $T(s, a, s')$ follows a Gaussian distribution $N(1.5, 0.25)$. The transition function $T(s, a, s')$ follows the uniform distribution $U(0, 1)$.

**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. 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 $\bigcirc - MDP$ 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:

\[
V(s) = \max_{a \in A_s} \pi(a) \left[ R(s, a) + \sum_{s'} T(s, a, s') V(s') \right].
\]
Online Baselines:
- LP-based approximation (LP-App) [12], where the online planning 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 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 $M_t$, 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 $M_t$. 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 an offline policy reuse method. In offline scenarios, the CMP is modeled as a 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


