Hierarchical Multi-Agent Reinforcement Learning for Allocating Guaranteed Display Ads

Lu Wang*
luwang@stu.ecnu.edu.cn
East China Normal University
Shanghai, China

Lei Han
leihan@tencent.com
Tencent, Inc.
Shenzhen, Guangzhou, China

Wei Zhang
zhangwei.thu2011@gmail.com
East China Normal University
Shanghai, China

Xinru Chen
xinruchen@tencent.com
Tencent, Inc.
Shenzhen, Guangzhou, China

Chengchang Li
chengchangli@tencent.com
Tencent, Inc.
Shenzhen, Guangzhou, China

Junzhou Huang
Junzhouhuang@tencent.com
Tencent, Inc.
Shenzhen, Guangzhou, China

Weinan Zhang
wnzhang@sjtu.edu.cn
Department of Computer Science
Engineering, Shanghai Jiao Tong
University
Shanghai, China

Xiaofeng He
xfhe@sei.ecnu.edu.cn
East China Normal University
Shanghai, China

Dijun Luo
dijunluo@tencent.com
Tencent, Inc.
Shenzhen, Guangzhou, China

ABSTRACT

In this paper, we study the problem of Guaranteed Display Ads (GDAs) allocation, which requires proactively allocate display ads to different impressions to fulfill their impression demands indicated in the contracts. Existing methods for this problem either assume the impressions are static or solely consider a specific ad's benefits. Thus it is hard to generalize to the industrial production scenario where the impressions are dynamical and large-scale, and the overall allocation optimality of all the considered GDAs is required. To bridge this gap, we formulate this problem as a sequential decision making problem in the scope of multi-agent reinforcement learning (MARL), by assigning an allocation agent to each ad and coordinating all the agents for allocating GDAs. The input are the states (e.g., the demands of the ad, the remain timesteps for displaying the ads) of each ad and the impressions at different timestep, and the output are the display ratios of each ads for each impression. Specifically, we propose a novel hierarchical multi-agent reinforcement learning (HMARL) method that creates hierarchies over the agent policies to handle a large number of ads and the dynamics of impressions. HMARL contains 1) a manager policy to navigate the agent to choose an appropriate sub-policy and 2) a set of sub-policies that let the agents perform diversely conditioning on their states. Extensive experiments on three real-world datasets from the Tencent advertising platform with tens of millions of records demonstrate significant improvements of HMARL over state-of-the-art approaches.

*Both authors contributed equally to this research.

SIGIR '21, July 11–15, 2021, Online
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

CCS CONCEPTS

• Information systems → Display advertising; **• Computing methodologies → Reinforcement learning**.

KEYWORDS

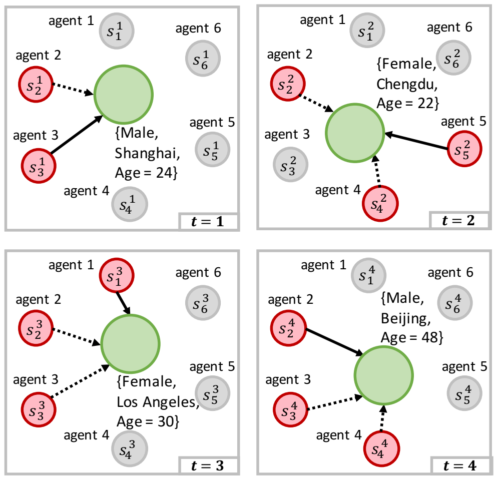
datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Lu Wang, Lei Han, Wei Zhang, Xinru Chen, Chengchang Li, Junzhou Huang, Weinan Zhang, Xiaofeng He, and Dijun Luo. 2021. Hierarchical Multi-Agent Reinforcement Learning for Allocating Guaranteed Display Ads. In *SIGIR '21: ACM SIGIR Conference on Research and Development in Information Retrieval, July 11–15, 2021, Online.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Advertising is an important industrial problem with a long-term pursuit in history. As reported in [1], the total cost of digital advertising on the World Wide Web is around $273B in 2018. There are two major display advertising paradigms: real-time bidding (RTB) [15, 37, 39] and guaranteed display ads (GDAs) [6, 8]. In RTB, an advertiser submits a bid in auctions happening in real time, while in GDAs, a contract is signed between an advertiser and an advertising platform in advance to ensure a certain amount of ad impressions to be displayed to some targeted populations (e.g., Female, Los Angeles and Age of 30). While both paradigms play indispensable roles in industries, the latter one has received much less research attention from the research field.

In this paper, we address the problem of GDAs allocation, which aims at allocating suitable display ads to an arrived impression. This problem is crucial and attracts massive attention from both



agent 1
agent 2
agent 3
agent 4
agent 5
agent 6
{Male, Shanghai, Age = 24}
{Female, Chengdu, Age = 22}
{Female, Los Angeles, Age = 30}
{Male, Beijing, Age = 48}
t = 1
t = 2
t = 3
t = 4

Figure 1: Allocation process of GDAs in an RL setting. Each green node indicates an impression with specific targeted populations and the other nodes indicate the allocation agents of ads. The ads eligible for the impression are marked as red and with arrows, where solid arrow means the ad obtains this impression.

industrial and academic communities [5, 18]. The existing studies in this respect fall into two main branches: static allocation optimization and heuristic methods. (1) Static allocation optimization aims to find an optimal allocation solution offline based on a simple assumption that the impressions are static [2, 11, 24, 26]. However, this assumption does not conform to the real industrial scenario that the impressions continuously arrive and exhibit dynamic distributions that are not exactly the same as the previous distribution. As such, the above methods are hard to generalize well online and dynamic impressions. (2) On the contrary, heuristic methods, such as High Water Mark (HWM) [8], SHALE [6], and proportional-integral-derivative (PID) controllers [4], consider the temporal dynamics of impressions to make real-time adaptation of allocation with considerable smoothness. Nevertheless, these methods solely focus on a single ad's benefits, which overlook the reality that the allocation strategy change of one ad might affect the optimal strategies of other ads.

Table 1 shows a case where ads might compete with each other. Assume there are two impressions (I1 and I2), both of which have 100 displays, and two ads (Ad1 and Ad2) with the impression demands of 100 and 80, respectively. We further suppose I1 is eligible for Ad1 and Ad2, which means the user tags of the impression match the ads' requirement, while I2 is only eligible for Ad2. When I1 first comes, PID controllers would give the same number (i.e., 50) of displays to Ad1 and Ad2. This is because PID controllers consider the profit of each ad separately. However, since I2 is not eligible for Ad1, PID controllers have to only give 30 displays to Ad2. As a result, the Ad1's demand for impression is not fulfilled. To achieve better overall optimality of all the considered GDAs, it is better to coordinate the allocation of ads.

Table 1: A case to show the competition between ads.

Impression	PID		Better allocation	
	Ad1	Ad2	Ad1	Ad2
I1	50	50	100	0
I2	0	30	0	80

In contrast to the above studies, we solve the problem of GDAs allocation with a game-theoretical method to make real-time adaption as well as coordinate the ads' strategies for global benefits. This is because: 1) The goal of allocating GDAs is to fulfill all ads' contracts. 2) Achieving the goal needs a series of allocation decisions for each ad. 3) As aforementioned, it is better to coordinate the decision of all ads. To achieve this, we formulate this problem as a multi-agent reinforcement learning problem[16, 21, 23, 27], wherein each ad (e.g., Ad1 and Ad2 in Table 1) is associated with an allocation agent and the satisfaction of demands is leveraged to quantify the reward. The series of actions generated by an agent corresponds to a Markov decision process (MDP). And the goal of the problem is to fulfill all the impression demands indicated in different contracts by maximizing the accumulated reward shown in Eq. 1. The details of the key components (e.g., agent, reward, etc.) w.r.t. multi-agent reinforcement learning will be introduced in Section 3.1.

More precisely, the agents make decisions to gain impressions at each time step. By evaluating the status of fulfilling the demands after each agent takes an action, the rewards for different agents could be determined. Therefore, meeting the demands of all the contracts is equivalent to solving this decision making problem among multiple agents. An illustration of the allocation process in the perspective of the MARL setting is shown in Fig. 1. For example, when an impression with one display arrives (e.g., t = 3), the agents (e.g., agent 1, agent 2, and agent 3) whose ads have the corresponding targeted populations (e.g., Female, Los Angeles, and Age = 30) take actions to compete for this impression to guarantee the ads' contracts. One of the ads (e.g., ad 1 with allocation agent 1) will be selected for the current display based on their actions and the eligible allocation agents will obtain reward signals conditioning on whether their ads get the impression. After that, the agents move into their new states based on the chosen actions.

Our study is large-scale and developed in the context of a realistic industry scenario, i.e., Tencent advertising platform, which serves over a hundred million active users. Modeling a large scale and varied amount of agents (about 5% to 25% variation w.r.t the total number of agents each day) allocation agents by MARL is difficult. Moreover, heterogeneity should be especially considered among the agents: (1) the agents have individual states and probably perform diversely, thus needing different policies. For example, some agents that have many eligible impressions in the morning need a policy taking more active actions in the morning; and (2) the relationship among some of the agents, either competitive or cooperative, can change over time. For instance, some ads with sufficient exposure will concede the impressions to other ads that are still not fulfilled.

To tackle the above challenges, we propose a novel Hierarchical Multi-Agent Reinforcement Learning (HMARL) method to create hierarchies on the agent policies. Specifically, the overall policy is decomposed into a manager policy and a collection of sub-policies, wherein the manager policy takes responsible for selecting a sub-policy for a given agent conditioning on its specific state. The agents that are assigned with the same sub-policy will share this sub-policy to determine their actions on impressions, making the parameter space largely reduced compared with independent agent learning. To stabilize the convergence, we utilize decentralized actors to execute agent actions and a centralized critic to coordinate the actions of multiple agents. We summarize our contributions as follows:

- To the best of our knowledge, this is the first study to treat the large scale allocation problem of GDAs as a multi-agent reinforcement learning problem, which enables to coordinate the allocation of ads to impression traffic and run parallel for large scale real-world advertisement platforms.
- We propose a hierarchical multi-agent reinforcement learning framework to deal with a large number of ads (agents) with heterogeneous states. The hierarchical framework clusters the agents into different sub-policies instead of learning individual policies. A coordinated Q-function is learned to coordinate the agents' strategies.
- We conduct extensive experiments on real-world datasets from the Tencent advertising platform. The results demonstrate that the proposed HMARL method significantly outperforms the existing methods for the allocation of GDAs, validating the benefits of introducing sub-policies in the hierarchical structure.

2 RELATED WORK

2.1 Allocation of GDAs

Previous work on allocation of GDAs can be summarized into two categories: static optimization based methods and heuristic methods. Static optimization based methods try to find a near-optimal solution by modeling this problem as a special version of static optimization on a bipartite graph [2, 11, 17–19, 24, 26, 30] via linear programming. These methods can find an optimal solution in a static environment. However, when considering real-world ad platforms, the distribution of impressions is highly dynamic due to the effects of both the advertisements and the consumers.

From a more practical view, [35] relies on the estimation of future impressions for each specific population to determine a serving rate of a target contract, which is too costly in industrial scenarios since there are many different populations (e.g., user tags). Inspired by this study, a more effective method HWM [8] just uses a greedy heuristic to determine the serving rate for each contract together with an allocation order. To achieve an approximate optimal allocation solution, SHALE [6] conducts an iterative algorithm by taking advantages of both the theory-oriented methods and practice-oriented methods.

Unfortunately, there are problems remained to be solved. They require an additional control step in online tests to adapt to the varied impression distributions, such as using feedback signals [8]. It incurs a very heavy computation burden due to a huge amount of impressions to obtain their allocation solutions. On the contrary, MARL sets the allocation actions for impressions based on the agents' states instead of the whole impression distribution. Thanks to the state transition in RL, the flexible adaptation of actions could be naturally achieved.

The advertising platform of Tencent, which contains a volume of tens of millions of users, chooses PID controllers [4] as the main allocation algorithm. Despite the practical advantages of this method as mentioned previously, it fails to model the communication and coordination among the considered ads, which could be important for the allocation problem whose goal is to meet the overall demands of the corresponding contracts.

2.2 Multi-Agent Reinforcement Learning

The goal of the MARL methods is to coordinate the agents to maximize the global team returns. The analysis of multi-agent systems has attracted great interest in both economic theory and artificial intelligence. The simplest approach for multi-agent systems is learning the agents independently [20, 23, 33, 38]. However, independent learning raises issues. This is because, from the perspective of any individual agent, the environment is non-stationary due to the effects of the other agents that act simultaneously. To overcome this problem, some existing methods have tried to infer other agents' policies and consider them in the Q function [21], or use importance sampling to correct bias in the experience replay [13]. Unluckily, the cases become worse when the amount of agents increases, and some agents have extremely heterogeneous behaviors, resulting in a more complex multi-agent task. That is, the agents have either competitive or cooperative relations with each other, and the relations can even vary over time. More recently, to deal with both competitive and cooperative agents, the MADDPG (multi-agent deep deterministic policy gradient) method [21] is developed, which contains a centralized critic and decentralized actors. And each actor learns its own policy. Nevertheless, with the growth of the agent number, the parameter size in MADDPG increases dramatically, making the method inappropriate for the problem of GDAs allocation considered here.

2.3 Hierarchical Reinforcement Learning

Long-term decision making is an important problem in traditional RL domains. The hierarchical reinforcement learning (HRL) is developed to decompose the long-term decision process into hierarchically structured short-term decision processes so that a policy can be organized along the hierarchy to manipulate an agent's behavior at multiple control levels. For example, some HRL methods use some domain knowledge to design a hierarchy over the actions [10, 28, 32] to reduce the search space. Another branch of HRL methods adopts option learning [10, 29, 32], which uses a two-layer hierarchy containing a manager policy to choose options and many sub-policies corresponding to different options. A typical example is the Feudal network [36] that learns a goal (option) embedding and computes some intrinsic rewards based on the goal to motivate the agent to act. Cao et al. [7] proposed a hierarchical critic to utilize the global information as additional reward signals. However, the above HRL methods focus on the single agent setting. A few approaches also study hierarchical policy for multi-agent settings to handle sparse and delayed rewards in the traditional RL domains [3, 14, 22, 25, 34]. Particularly, inspired by MAXQ [10], some studies [14, 22, 25] propose to share sub-task value functions

Table 2: Notation descriptions.

Notation	Description
$s_t \in \mathbb{R}^4$	lThe local state which consist
the n-th agent's observed features.	
$\bar{s}_t^n \in \mathbb{R}^N$	lThe interaction state of the r
indicates the similarity with the N ads.	
$\hat{s}_t^n = [\bar{s}_t^n, s_t^n]$	The integrated state.
$\hat{r}_t^n \in \mathbb{R}$	The reward of the manager p
$r_t^n \in \mathbb{R}$	The reward of the sub-policy.
$\hat{\mu}_{\hat{\theta}}$	The manager policy with par
μ_{θ_k}	The k-th sub-policy with para
$\hat{a}_t^n$	The action generated by man
a_t^n	The action generated by sub-
$Q(s_t^n, a_t^n)$	$Q(s_t^n, a_t^n) = \sum_{t'=t}^{T} \mathbb{E}[r(s_t^n, a_t^n)$
$\hat{Q}_{\hat{w}}$	The Q function of the manag
Q_{w_k}	The Q function of the sub-po
ϵ	The similarity threshold.

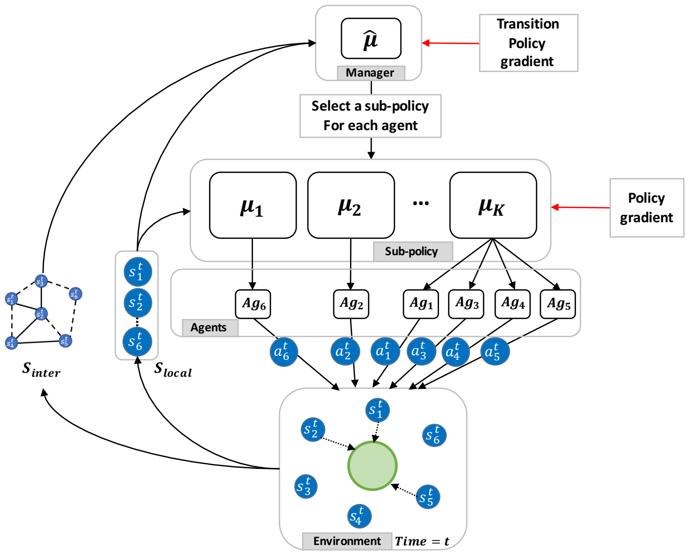

$\hat{\mu}$
Manager
Transition Policy gradient
Select a sub-policy For each agent
μ_1
μ_2
…
μ_K
Sub-policy
Policy gradient
Ag_6
Ag_2
Ag_1
Ag_3
Ag_4
Ag_5
Agents
a_6^t a_2^t a_1^t a_3^t a_4^t a_5^t
s_1^t s_2^t s_3^t s_4^t s_5^t s_6^t
S_{inter}
S_{local}
Environment
$Time = t$

Figure 2: Overview of the HMARL method.

among agents, which require to pre-define the sub-task and focus on discrete actions. Recently, neural network based hierarchical multi-agent models are proposed [3, 34] to assign each agent with a policy in a toy environment, not applicable for a large-scale environment.

In this work, we propose a new hierarchical MARL method and introduce it to a fresh domain, i.e., the allocation of GDAs, where the considered number of agents is very large and the agents are heterogeneous with continuous action spaces.

3 THE HMARL METHOD

In this section, we first formulate the allocation of GDAs as an MDP problem with detailed definitions for agent, state space, action space, and reward design, etc. Then, we elaborate on the proposed HMARL method. Finally, we explain the training processes for the manager policy and sub-policies, respectively.

3.1 Problem Formulation

We first formulate allocating GDAs as a multi-agent problem or Markov game [12], where there are N agents on behalf of merchants to allocate N ads to impressions. A multi-agent problem is defined by a set of states $\{\mathcal{S}_1, ..., \mathcal{S}_N\}$ describing the statuses of all display agents, $\mathcal{A}_n$ indicates the action space of agent n. An action $a \in \mathcal{A}_n$ denotes the adjustment ratio for display. According to state s_t^n of agent n in the t-th time step, a policy $\mu : \mathcal{S} \longrightarrow \mathcal{A}$ is to determine a specific action a_t^n. After each agent takes an action, its state is transferred to the next one followed by obtaining a reward r_t^n based on a function of the state and all agents' actions. The initial states are determined by a predefined distribution. Agent n aims to maximize its own total expected return $R_n = \sum_{t=1}^{T} \gamma_{t-1} r_t^n$ where γ is a discount factor and T is a time horizon. We describe the details of agent and policy in our setting as follows:

- **Agent**: In GDAs, we consider each ad with an allocation agent. Since assigning each agent with a unique policy is computationally expensive when the number of ads is large, we consider K allocation sub-policies which are shared among N agents, where $K \ll N$. The agents should not only fulfill their own demands but also coordinate with other agents to satisfy the overall demands of all contracts from the perspective of an advertising platform. These agents act in a mixture way that can be both cooperative and competitive in different time steps.
- **Hierarchical Sharing Policy:** The proposed HMARL framework consists of a manager policy and a collection of sub-policies, whose architecture is shown in Fig. 2. Given an impression, HMARL works by first using a manager policy to select a sub-policy for each ad that is eligible for the impression based on the state of the ad. Consequently, it leverages the selected sub-policy to perform a specific allocation action.

Given an agent, we take the impression that comes at time step t and the n-th ad as examples to explain state, action, and reward function involved in HMARL as follows:

- **State**: The agent of the n-th ad has an interaction state $\bar{s}_t^n$ which indicates the global relationship with the other agents and a local state s_t^n used to describe its situation. We denote $\hat{s}_t^n = [\bar{s}_t^n, s_t^n]$ as the integrated state. The interaction state $\bar{s}_t^n$ is represented as an N-dimensional binary vector to indicate the relations between the current agent and the others. To calculate $\bar{s}_t^n$, we first compute the Jaccard similarity between the n-th ad and the other ads. In particular, if the Jaccard similarity between the n-th ad and the m-th ad is larger than a pre-defined threshold ϵ, we set the m-th value of $\bar{s}_t$ as 1. Otherwise, it is set to 0.
The local state is defined as a 4-dimensional vector, where $s_t^n = [w^n, c_t^n, l_t^n, a_{t-1}^n]$:
(1) w^n ($w^n = d^n/S^n$) presents the degree of resource shortage for the ad, where d^n is its demand and S^n is the amount of its eligible impressions. If the ad is a warm-start one, we use its latest estimated eligible amount of impressions as S^n. Otherwise, we use the amount of the ad most similar to S^n instead. Note

that this is essentially different from forecasting the impression distribution for each targeted population, which is fine-grained, costly, and quite error-prone.
(2) c_t^n ($c_t^n = t/T$) denotes the ratio of how much time has passed compared to the valid time horizon of the contract.
(3) l_t^n ($l_t^n = \frac{\sum_{t'=1}^{t} e_{t'}^n)}{d^n}$ presents the ratio of accumulated displays divided by the demand until time step t, where $e_{t'}^n$ is the number of displays for the ad at time t'.
(4) a_{t-1}^n denotes the value of allocation action at time step $t-1$.

- **Action**: There are two categories of actions in HMARL. The high-level action is represented as $\hat{a}_t^n = \hat{\mu}_{\hat{\theta}}(\hat{s}_t^n)$. It is performed by the manager policy $\hat{\mu}_{\hat{\theta}}$ and denotes the possibility of the sub-policies to be selected for the ad. The low-level action for sub-policy μ_{θ_k} is denoted as $a_t^n = \mu_{\theta_k}(s_t)$, which takes the range $[0, 1]$.
To convert the action value to a real exposure ratio, we follow the previous work [8]. Given an impression, we first rank the eligible ads by their action values in descending order. Then the first ad is assigned with an exposure ratio $e_t^1 = a_t^1$, and the j-th ad is assigned with an exposure ratio e_t^j, where $e_t^j = (1 - \sum_{w=1}^{j-1} e_t^w)a_t^j$. Afterwards, we follow the order and utilize the exposure ratios to determine the allocation of ads. For example, if the j-th ad is not displayed, we continue to determine whether to display the $(j+1)$-th ad by sampling according to its ratio. If none of the ads get this impression, it will be used for advertising auctions.
- **Reward**: $\hat{r}_t^n = \hat{\mathcal{R}}(\hat{s}_t^n, \hat{a}_t^n)$ and $r_t^n = \mathcal{R}(s_t^n, a_t^n)$ are two kinds of reward functions for manager policy and sub-policies, respectively. In particular, the manager reward is defined as the summation of the reward of the corresponding sub-policy it chooses during the V time-steps: $\hat{r}_t^n = \sum_{i=t}^{t+V}(r_i^n)$. Then we formulate the reward function for sub-policies, which includes three parts:
(1) Self-oriented reward: $-(a_t^n - (1 - l_t^n))^2 + 0.5$. It considers each individual benefit. When the ratio of the accumulated exposure is small (e.g., $l_t^n = 0$), the agent with a larger action value will obtain a larger reward. Conversely, when the exposure ratio is large (e.g., $l_t^n = 1$), the agent with a larger action will obtain a smaller reward. 0.5 is an offset to keep the reward in a reasonable range.
(2) Coordination-oriented reward: $-(a_t^n - \frac{(1-l_t^n)}{\sum_{i=1}^{N}(1-l_t^i)})^2 + 0.5$. This reward tries to coordinate the agents' actions by matching the action with the fraction between the agent's remaining exposure ratio and the summation of the other agents' remaining exposure ratios.
(3) Temporal reward:

$$r_t^n = \begin{cases} \lambda_z * \omega & \text{if } l_t^n \in (0.95 * \omega, 1.05 * \omega) \\ (1.05 - l_t^n) * \lambda_o * \omega & \text{if } l_t^n >= 1.05 * \omega \\ -\lambda_{u1} * \omega & \text{if } l_t^n < 0.5 * \omega \\ -\lambda_{u2} * \omega & \text{if } l_t^n <= 0.95 * \omega \end{cases}$$

where 0.95 and 1.05 are slack thresholds of under-delivery and over-delivery exposures because the exact demand of the contract is hard to satisfy. $\lambda_z, \lambda_o, \lambda_{u1}, \lambda_{u2}$ are manual factors in the range $[0, 1]$ to control the relative influence of normal-delivery, over-delivery and under-delivery exposures. ω takes the form $\omega = \frac{t}{T}$. This reward also tries to improve the global benefits. In addition, temporal reward evaluating the agents regularly helps smooth the action values of the agents. The effectiveness of these reward parts will be shown in Fig. 5.

3.2 HMARL: The Proposed Method

Our hierarchical architecture is illustrated in Fig. 2. There are two types of policies. The manager policy decides to choose a sub-policy based on current observation with the interval of V time steps, and the sub-policy picks a primitive action at each time step. The objective function of HMARL is to maximize the cumulative rewards of both the manager policy and the sub-policies to achieve overall satisfaction.

As the example shown in Table 1, our goal is to allocate the two ads to the two impressions to satisfy their demands. The rewards indicate the satisfaction ratios of the ads at each time step. It is intuitive to maximize the cumulative reward in the long run instead of the immediate reward $\hat{r}_t^n$ for the n-th agent because sacrificing some display chances of some ads is beneficial for fulfilling the other ads' demands which are more urgently needed. With this intuition, we define the following objective function to be maximized:

$$J(\hat{\theta}, \theta_k)_{k\in\{1,\ldots,K\}} = J(\hat{\theta}) + \sum_{k=1}^{K} J(\theta_k)\,, \tag{1}$$

where $J(\hat{\theta})$ is the objective function of the manager policy. It tries to maximize the cumulative rewards over all agents, which is defined as follows:

$$J(\hat{\theta}) = \sum_{n=1}^{N} \mathbb{E}_{\hat{s}_t^n, \hat{a}_t^n \sim \hat{\mu}_{\hat{\theta}}} [\sum_{t\in\Upsilon} \gamma^{\lfloor t/V \rfloor} \hat{r}_t^n(\hat{s}_t^n, \hat{a}_t^n)] \tag{2}$$

where $\Upsilon = \{1, 1+V, 1+2V, \cdots, 1+\lfloor T/V \rfloor \cdot V\}$, $\hat{\mu}_{\hat{\theta}}$ is the manager policy, and $\gamma \in [0, 1]$ is a discount factor to determine the importance of future rewards. Its goal is to select a sequence of sub-policies for each ad to fulfill its contract. $J(\theta_k)$ is the objective function of the sub-policy k, whose goal is to output a display ratio to compete for an impression. Specifically, the objective function is to maximize the cumulative rewards received in T time steps, which is given by:

$$J(\theta_k) = \sum_{n=1}^{N} \mathbb{E}_{s_t^n, a_t^n \sim \mu_{\theta_k}(k=\arg\max \hat{a}_t^n)} [\sum_{t=1}^{T} \gamma^{t-1} r_t^n(s_t^n, a_t^n)] \tag{3}$$

The parameters $\hat{\theta}$ and $\theta_k, \{k = 1, .., K\}$ will be trained based on policy gradient [36], wherein the gradient of the objective has the general form (the subscript n and k are omitted) as follows:

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\mu_\theta(\tau)}[\nabla_\theta \log \mu_\theta(\tau) r(\tau)] \\ &= \mathbb{E}_{s_t, a_t \sim \mu_\theta}[\sum_{t=1}^{T} \nabla_\theta \log \mu_\theta(a_t|s_t) \sum_{t'=t}^{T} r(s_{t'}, a_{t'})]\,. \end{aligned} \tag{4}$$

With the theorem of deterministic policy gradient [31], we introduce the Q function, defined as $Q(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}[r(s_{t'}, a_{t'})|_{a_{t'}=\mu(s_{t'})}]$, to replace $\sum_{t'=t}^{T} r(s_{t'}, a_{t'})$ for lower variance. In this paper, we learn an approximate Q function with parameter w instead of directly calculating it from the samples. Then Eq. **??** is rewritten as:

$$\begin{aligned} \nabla_\theta J(\theta) = \mathbb{E}_{s_t, a_t \sim \mu_\theta}[\sum_{t=1}^{T} \nabla_\theta \log \mu_\theta(a_t|s_t) \\ \cdot \nabla_{a_t} Q_w(s_t, a_t)|_{a_t=\mu_\theta(s_t)}]\,. \end{aligned} \tag{5}$$

In the following, we elaborate the details of manager policy learning and sub-policy learning.

3.2.1 Manager Policy Learning. The manager policy in HMARL produces a higher-level action $\hat{a}_t^n$ to choose a desired sub-policy for the n-th agent based on its interaction state $\bar{s}_t^n$ and local state s_t^n in every V time steps, where V is a hyperparameter to be tuned. We define a multi-step transition probability function $p(\hat{s}_{t+V}^n \mid \hat{s}_t^n, \hat{a}_t^n)$ for the manager. It denotes the probability that action $\hat{a}_t^n = \hat{\mu}_{\hat{\theta}}(\hat{s}_t^n)$ will cause the system to transform from state $\hat{s}_t^n$ to state $\hat{s}_{t+V}^n$ in V time steps. From time step t to $t+V$, the action $\hat{a}_t^n$ will not be changed. After V time steps, the manager takes another action $\hat{a}_{t+V}^n$.

The manager policy stores its experience in buffer $\hat{\mathscr{D}}(\hat{s}_t^n, \hat{a}_t^n, \hat{r}_t^n, \hat{s}_{t+V}^n)$, where $\hat{r}_t^n$ is the reward obtained by the manager policy after choosing a sub-policy. More concretely, $\hat{r}_t^n$ is calculated by $\hat{r}_t^n = \sum_{t'=t}^{t+V}(r_{t'}^n)$. Consequently, the gradient of the manager policy is formulated as follows:

$$\nabla_{\hat{\theta}} J(\hat{\theta}) = \sum_{n=1}^{N} \mathbb{E}_{\hat{s}_t^n, \hat{a}_t^n \sim \hat{\mu}_{\hat{\theta}}} [\sum_{t \in \Upsilon} \nabla_{\hat{\theta}} \hat{\mu}_{\hat{\theta}}(\hat{s}_t^n) \nabla_{\hat{a}^n} \hat{Q}_{\hat{w}}(\hat{s}_t^n, \hat{\mu}_{\hat{\theta}}(\hat{s}_t^n))] , \quad (6)$$

where the Q-value function $\hat{Q}$ w.r.t. time step t is obtained by minimizing the following loss:

$$\begin{aligned} \mathcal{L}(\hat{w}) = \sum_{n=1}^{N} \mathbb{E}_{\hat{s}_t^n, \hat{a}_t^n, \hat{r}_t^n, \hat{s}_{t+V}^n \sim \hat{\mu}_{\hat{\theta}}} [(\hat{Q}_{\hat{w}}(\hat{s}_t^n, \hat{a}_t^n) - \hat{r}_t^n(\hat{s}_t^n, \hat{a}_t^n) - \\ \gamma \hat{Q}_{\hat{w}}(\hat{s}_{t+V}, \hat{a}_{t+V}))^2)\}] . \end{aligned} \quad (7)$$

3.2.2 Sub-Policy Learning. Inspired by MADDPG [21], we use decentralized actors and a centralized critic to handle mixed cooperative and competitive environments, where the centralized critic is used to capture the relations among agents and the decentralized actors enable to perform actions in parallel. As for sub-policy μ_{θ_k}, its Q function is given as $Q_{w_k}(s_t, s_t^-, a_t, a_t^-)$, which is a centralized action-value function taking all the actions and states of the agents as input. $\boldsymbol{\mu} = \{\mu_{\theta_1}, \mu_{\theta_2}, ..., \mu_{\theta_K}\}$ is a set of sub-policies. s_t^- and a_t^- indicate the average state embedding and the average action of the agents in except the target agent n, which are given by:

$$s_t^- = \frac{1}{N-1} \sum_{j \neq n}^{N} s_t^j, \quad a_t^- = \frac{1}{N-1} \sum_{j \neq n}^{N} a_t^j . \quad (8)$$

The centralized action-value function $Q_{w_k}^{\boldsymbol{\mu}}$ is updated based on the following loss:

$$\begin{aligned} \mathcal{L}(w_k) = \sum_{n=1}^{N} & \mathbb{E}_{s_t^n, a_t^n \sim \mu_{\theta_k} (k = \arg\max \hat{a}_t^n)} \\ & [(Q_{w_k}(s_t^n, s_t^-, a_t^n, a_t^-) - y)^2] \\ y = r_t & + \gamma Q_{w_k}(s_{t+1}^n, s_{t+1}^-, a_{t+1}^n, a_{t+1}^-) \\ a_{t+1}^n & = \mu_{\theta_k}(s_{t+1}^n), a_{t+1}^- = \mu_{\theta_k}(s_{t+1}^-) , \end{aligned} \quad (9)$$

where the input of the Q function are the states and actions of all the agents. This is inspired by MADDPG, where the centralized critic with deterministic policies works well in practice. The reason would be that if we know the actions taken by all the agents, the environment is stationary even as the policies change. We use an alternative gradient descent method to update the Q functions, where Q_{w_k} is mutually updated by fixing the other $K-1$ sub-policies' parameters.

As for optimizing sub-policy μ_k, the gradient of the expected return is derived as follows:

$$\begin{aligned} \nabla_{\theta_k} J(\theta_k) = \sum_{n=1}^{N} \mathbb{E}_{s_t^n, a_t^n \sim \mu_{\theta_k} (k = \arg\max \hat{a}_t^n)} [\nabla_{\theta_k} \log \mu_{\theta_k}(s_t^n) \\ \nabla_{a_t^n} Q_{w_k}(s_t^n, s_t^-, a_t^n, a_t^-) \mid_{a_t^n = \mu_{\theta_k}(s_t)^n, a_t^- = \mu_{\theta_k}(s_t^-)}] \end{aligned} \quad (10)$$

Similar to the previous optimization for Q-functions, the alternative gradient decent approach adopted for optimizing θ_k keeps the other policies' parameters fixed to ensure a more stable training process.

Algorithm 1 HMARL for Allocation of GDAs

```
Require: Number of sub-policies $K$, number of contracts $N$, reward function $r_t^n$ of the environment, time span $V$, total time steps $T$, and number of epochs $U$.
 1: Warm-starting manager policy $\hat{\mu}_{\hat{\theta}}$, sub-policies $\mu_{\theta_k}$, Q-functions $\hat{Q}$ and $Q_{w_k}$, and replay buffers $\hat{\mathscr{D}}$ and $\mathscr{D}_k$;
 2: for $epochs = 0$ to $U$ do
 3:     clear $\hat{\mathscr{D}}$ and $\mathscr{D}_k$ ;
 4:     for $t = 1$ to $T$ do
 5:         Sample a tuple $(\hat{s}_t^n, \hat{a}_t^n, \hat{s}_{t+V}^n, \hat{r}_t^n)$ using manager policy $\hat{\mu}_{\hat{\theta}}$.
 6:         Add $(\hat{s}_t^n, \hat{a}_t^n, \hat{s}_{t+V}^n, \hat{r}_t^n)$ into $\hat{\mathscr{D}}$ every $V$ time steps ($n = \{1, ..., N\}$).
 7:         Sample a tuple $(s_t^n, a_t^n, s_{t+1}^n, r_t^n)$ using sub-policy $\mu_{\theta_k}$.
 8:         Add a tuple $(s_t^n, a_t^n, s_{t+1}^n, r_t^n)$ into $\mathscr{D}_k$ ($k = \{1, ..., K\}$).
 9:         Update manager policy $\hat{\mu}_{\hat{\theta}}$ and Q-function $\hat{Q}$ via $\hat{\mathscr{D}}$.
10:         Update sub-policies $\mu_{\theta_k}$ and Q-function $Q_{w_k}$ via $\mathscr{D}_k$ ($k = \{1, ..., K\}$).
11:     end for
12: end for
```

3.3 Training Algorithm

Algorithm 1 shows the training process of the HMARL method. To facilitate the training, we first pre-train the manager policy and the sub-policies to provide a warm-start of the parameters. That is, we first cluster the agents or ads into K groups, followed by training the K groups of agents in K sub-policies via DDPG [9]. At the same time, the manager policy $\hat{\mu}_{\hat{\theta}}$ is pre-trained by classifying the agents into K groups through a simple supervised learning scheme.

In the training stage, we collect the trajectories of the manager policy and sub-policies. At each time step, we update our model by sampling tuples from the buffers. Due to the centralized Q-networks, our method requires to obtain states and actions of all agents at the same time, and we simultaneously save them to the buffers. However, for the testing step, only the decentralized policy networks are used.

Computational complexity: As shown in Algorithm 1, the time complexity of our method is $O(UTN(d+\sum_{c=2}^{C} d_{c-1}d_c))$, where T is the number of steps or the number of impressions. d is the dimension of state, C is the number of layers in the neural network, and d_c is the dimension of the c-th layer in the actor and critic network.

Table 3: Basic statistics of the real world datasets from Tencent used in this paper.

Log date	#Contract	#Impression	#Demand
16/04/2018	134	2,516,251	137,230
08/11/2018-10/11/2018	716	14,805,173	1,989,820
27/12/2018-03/01/2019	1,727	45,208,115	8,971,240

4 EXPERIMENTS

4.1 Datasets

To validate the performance of the proposed method, we study three large scale real-world datasets from the Tencent advertising platform, where we use the actual logs of ads and impressions. Table 3 shows the statistics of the datasets collected from different time periods. The first dataset covers a one-day time span, and the other two datasets are collected using three-day and one-week time spans, respectively. Following the previous studies [6, 8], the three datasets were down-sampled with a rate of $1/512$. For the obtained datasets, we used the corresponding active GDAs contracts with different demands.

As aforementioned, about 5% to 25% of the ads will be updated every day. The historical log data is used to estimate the total supply amount of each contract. We segmented the datasets into training and testing parts. Specifically, for the first one-day dataset, we used the first 12-hour log data for training and the rest 12-hour log data for testing. And for the two datasets spanning over multiple days, we utilized the logs from a previous day to train the model and the logs of a consequent day for testing. To enable to tune some hyper-parameters of different methods, we chose a 10% subset from each train set as the validation set.

In RL, an agent needs to interact with an environment. Luckily, the GDAs provide a natural environment for the agents to interact. During training, given the historical log data and the contracts of the GDAs, we trained our policies by going through the historical log impressions with several episodes. During each episode, the policy can be evaluated by measuring whether it fulfills the demands of the contracts to obtain the reward signal. As for testing, we utilized the learned policy to act in a new day's impression logs and contracts. As such, the offline testing is regarded as an effective way to evaluate the allocation policy of GDAs [6, 8]. The codes and desensitized data will be released as the publication of this paper.

4.2 Evaluation Metrics

We chose the performance metrics, i.e., under-delivery rate, normal-delivery rate, and over-delivery rate, that reflect the interests of the industrial world to evaluate the different methods described in Section 4.3. 0.95 and 1.05 were chosen to be slack thresholds of under-delivery and over-delivery rates because an exact demand of the contract is hard to be satisfied.

1. **Under-delivery Rate**: This represents the ratio between the number of under-delivery contracts and the number of total contracts, i.e., $UR = \frac{\sum_{n=1}^{N} I_n}{N}$, where I_n is an indicator function. If $\sum_{t=1}^{T} e_t^n < 0.95 * d^n$, $I_n = 1$, and otherwise, $I_n = 0$.
2. **Normal-delivery Rate**: This represents the amount of normal-delivery contracts accounting for the amount of contracts, i.e., $NR = \frac{\sum_{n=1}^{N} I_n}{N}$. If $0.95 * d^n \leq \sum_{t=1}^{T} e_t^n \leq 1.05 * d^n$, $I_n = 1$. Otherwise, $I_n = 0$.
3. **Over-delivery Rate**: This represents the amount of over-delivery contracts accounting for the amount of contracts, i.e., $OR = 1 - (UR + NR)$.

In practice, UR can be considered as the most important metric. This is because if some contracts are not fulfilled, the advertising platform will make compensation for the advertisers. OR is also important, denoting the waste of impressions which causes revenue loss of the advertising platform.

4.3 Delivery Policies for Comparison

All the adopted methods for comparison in this paper are as follows:

- **Random**: This method applies a random policy.
- **Static Action**: This method calculates the allocation fraction w^n ($w^n = d^n/S^n$) as a static action of each agent.
- **DG, SG**: Both DG and SG give a higher priority to the contracts that are difficult to fulfill. In particular, DG gives the current agent whose ad has the highest demand with $action = 1$ and gives $action = 0$ for the other agents. SG gives $action = 1$ for the current agent whose ad has the lowest supply and sets the other agents' actions to 0.
- **HWM [8]**: HWM uses a greedy heuristic to determine a serving rate as the action for each contract together with an allocation order.
- **PID [4]**: This approach is deployed on the Tencent advertising platform. PID designs a minute-level goal for each ad and makes the ads adjust their actions to fulfill the goals by capturing the state variation of ads.
- **SHALE [6]**: SHALE extends HWM to incorporate dual solutions. It first finds reasonable dual solutions with an iterative algorithm, followed by converting the reasonable set of dual solutions into a primal solution.
- **LR**: It trains a linear regression model by taking the features of PID as input and the corresponding actions of PID as the labels.
- **RFR**: RFR replaces linear regression with random forest to train a policy with the same samples.
- **FeUdal [36]**: FeUdal is a classical HRL model consisting of a manager and a worker. The manager takes a high-level action to generate a goal embedding to guide the worker to take a low-level action.
- **Flat_HMARL**: This is the variant of HMARL which is with only one sub-policy and without the manager policy.
- **HMARL**: This is the approach proposed in this paper.

4.4 Implementation Details

In this part, we clarify some implementation details of our proposed approach as follows:

- The manager was trained by DDPG, which contains an actor network and a critic network (i.e., the Q function). The actor network has three layers, with the size of 6-32-32-4 and activation functions of relu-relu-softmax, where 6 corresponds to the dimension of state $\hat{s}_t^n$ shown in Section 3. The

Table 4: Performance comparison on the three datasets with different periods for allocation of GDAs.

	27/12/2018-03/01/2019			08/11/2018-11/11/2018			16/04/2018		
Method	UR	NR	OR	UR	NR	OR	UR	NR	OR
Random	0.442	0.019	0.537	0.190	0.042	0.770	0.313	0.030	0.642
Static Action	0.728	0.090	0.180	0.913	0.086	**0**	0.813	0.171	**0**
DG	0.471	0.508	0.019	0.388	0.564	0.051	0.485	0.500	0
SG	0.388	0.601	**0.009**	0.526	0.435	0.043	0.724	0.261	0
PID	0.218	0.760	0.020	0.112	0.806	0.086	0.224	0.634	0.127
HWM	0.431	0.492	0.075	0.281	0.637	0.097	0.366	0.470	0.149
SHALE	0.502	0.472	0.024	0.221	0.702	0.087	0.244	0.606	0.134
LR	0.295	0.683	0.021	0.272	0.518	0.220	0.388	0.448	0.149
RFR	0.272	0.694	0.039	0.250	0.629	0.119	0.366	0.590	0.030
FeUdal	0.255	0.692	0.049	0.174	0.714	0.111	0.299	0.612	0.075
Flat_HMARL	0.237	0.716	0.055	0.142	0.761	0.098	0.225	0.647	0.139
HMARL	**0.138**	**0.835**	0.022	**0.058**	**0.875**	0.073	**0.089**	**0.813**	0.082

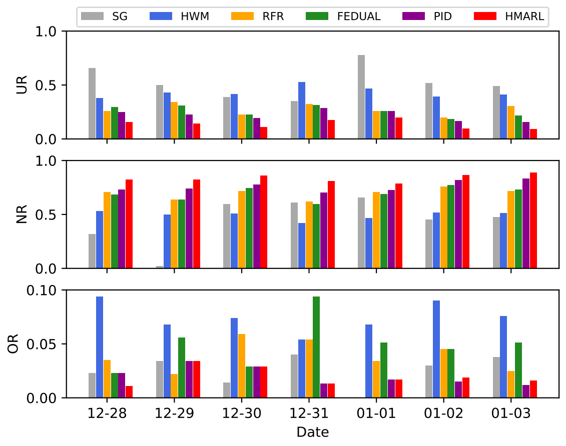


Figure 3: Performance of compared methods on different days (x-axis).

Q function also has three layers, with the size of 4-32-32-1 and activation functions of relu-relu-linear.

- The number of sub-policies was tuned to be 4. Each sub-policy is learned by DDPG as well.
- The actor network of each sub-policy is a three-layer neural network (i.e., with the size of 4-32-32-1 and active functions relu-relu-sigmoid).
- The Q function of each sub-policy is a three-layer neural network (i.e., with the size 4-32-32-1 and active functions of relu-relu-linear).
- HMARL was optimized by Adam, with the learning rate 6e-04 for both the manager network and the actor networks of sub-policies, and 6e-05 on the critic network.
- We tuned the discount factor γ among {0.1, 0.3, 0.7, 0.9, 0.93, 0.95, 0.97, 0.99, 1} and obtained the corresponding NR, i.e., {0.809, 0.812, 0.804, 0.827, 0.833, 0.843, 0.837, 0.853, 0.844} of one validation dataset. The best result is 0.853 and thus we set γ to 0.99.
- We tuned the time span V to be 5 for its better performance on validation datasets.

4.5 Performance Comparison

Their different distributions of the impressions and contracts ensure the comparison to be reliable and robust. Table 4 shows the results of method comparison on the three datasets that have different distributions of the impressions and contracts to ensure reliability. Based on the results, we observe that:

(1) HMARL consistently outperforms the other methods on UR and NR in different impression distributions. The reasons are: i) HMARL captures the dynamic change of the impression and contract states to enable the adaptive and sequential decisions based on its policy function, without relying on the specific impression distribution (compared with HWM and SHALE); ii) HMARL regards the allocation of GDAs as a multi-step decision process, reflecting the real-world practice (compared with LR and RFR); iii) instead of considering the individual interest of each agent, HMARL also considers the global benefits by coordinating the actions of all the agents (compared with PID); iv) HMARL utilizes different sub-policies to handle the heterogeneous states of agents (compared with FeUdal and Flat_HMARL). However, HMARL does not obtain the smallest over-delivery rate, as compared to the lower over-delivery rates achieved by the conservative methods SG and Static Action. The reason might be that many data examples in the training process are under the condition of under-delivery rates and thus the policy is optimized to take large actions.

(2) PID achieves good performance by dynamically capturing the change of the environment and making adaptive decisions, which also does not rely on the estimation of impression distributions. However, compared with HMARL, PID ignores the coordination among the agents.

(3) Reinforcement learning methods (FeUdal and HMARL) have better performance than supervised learning (SL) methods, possibly because the slight change of the action generated by SL may cause a "drift" sequence compared with the original sequence.

(4) Both HWM [8] and SHALE [6] require an accurate impression distribution estimation which might cause poor performance due to the dynamics of the impression distribution.

(5) Although FeUdal ensures the assumption of the Markov decision process, a monolithic policy network is hard to handle the heterogeneous states of the agents. We observe that agents in FeUdal tend to take similar actions.

(6) Static Action, DG, and SG show very low performance on UR and NR. This is attributed to the fact that these methods neither consider the dynamic states of the ads nor the coordination for social benefits. Surprisingly, these conservative policies help these methods achieve the lowest OR by taking slightly changing actions. The reason is that they always take a small delivery rate.

In addition, we report the detailed performance on each day of the one-week dataset in Fig. 3. Due to the varied impression distribution of each day, all the results of the compared methods are changed. However, HMARL also consistently outperforms the other methods in terms of UR and NR and achieves a slightly larger OR.

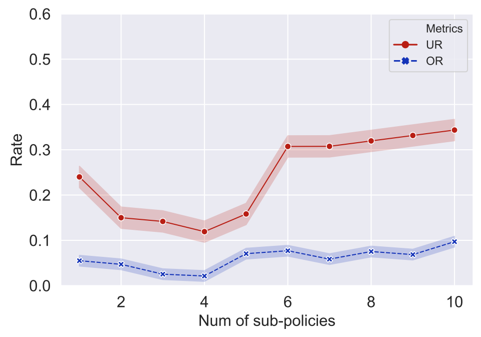


Figure 4: Variation of the UR and OR rates w.r.t. the number of sub-policies.

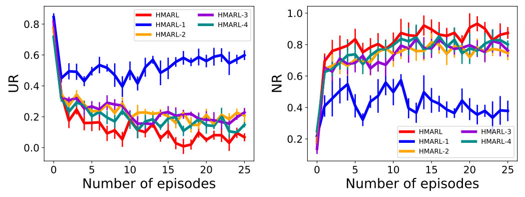


Figure 5: Convergence and performance comparison of HMARL in its variations without different kind of rewards.

4.6 Impact of Different Numbers of Sub-Policies

It is of great interest to investigate how different numbers of sub-policies affects the performance of HMARL. Specifically, we test the number from the set of $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and show their performances in Fig. 4. First of all, when the number is set to 4, HMARL achieves the best performance. From the perspective of the overall trend, the performance improves as the number of sub-policies increases from 1 to 4. This is intuitive since it is necessary to introduce more agents to handle complex heterogeneous states. However, when the number of sub-policies exceeds 4, the performance of HMARL suffers from a dramatic drop. The reason might be that it is more challenging to ensure the good convergence of HMARL and guarantee a rational decision made by the manager policy with more sub-polices.

4.7 Impact of Different Reward Functions

Reward design plays a crucial role in reinforcement learning. In this paper, we verify four kinds of reward functions mentioned in Section 3.1: 1) self-oriented reward, 2) coordination-oriented reward, 3) temporal reward, and 4) global reward. To evaluate the effectiveness of different rewards, we each time remove one of these 4 reward functions in the same order as above and denote the corresponding methods as *HMARL-1*, *HMARL-2*, *HMARL-3*, and *HMARL-4*, respectively. As we can see in Fig. 5: (1) All of the four reward functions make contributions to improving the performance of HMARL and promise a convergence. The reason is that, in this mixed cooperation and competition task, both self-interest and cooperation rewards are required for agents to adjust their actions while these cooperating agents encounter conflicts. (2) Self-oriented reward significantly boosts the convergence and performance of HMARL. (3) Coordination-oriented reward, temporal reward, and global reward make nearly equal contributions to improving the convergence and performance of HMARL. Noting that we consider the convergence of UR or NR during training as the stopping criteria. Specifically, if the difference of UR or NR in two consecutive steps is smaller than 0.02, we regard the training process converges.

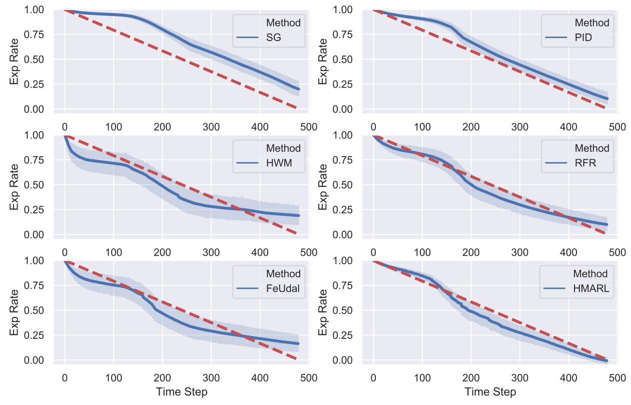


Figure 6: Visualization of Smoothness.

Table 5: Quantitative Comparison of Smoothness.

Method	σ^{25}	σ^{50}	σ^{75}	σ^{95}
SG	0.079	0.187	0.301	0.443
PID	-0.021	0.052	0.171	0.355
HWM	-0.533	-0.206	0.360	0.476
RFR	-0.522	-0.094	0.212	0.417
FeUdal	-0.529	-0.227	0.224	0.417
HMARL	-0.133	0.001	0.101	0.477

4.8 Smoothness Study

In the real world, ad companies desire a smooth exposure as time goes on, instead of delivering all the demands only focusing on several hours during a day. This motivates us to compare the *smoothness* of these methods. Intuitively, a perfect smoothness exposure for a one-day contract display 1/24 demands in every hour. If an allocation method behaves similarly to the perfect smoothness exposure, it might better comply with the expectation of companies. As shown in Fig. 6, the red dashed line is the perfect smoothness line and the y-axis indicates the average cumulative under-exposure ratio of each ad. We find that none of the compared methods are the same as the perfect smoothness line. Among them, HWM and FeUdal are wavier than other methods. PID and HMARL behave similarly as the perfect line, but at the end of the time steps, HMARL has a smaller under-exposure ratio than PID.

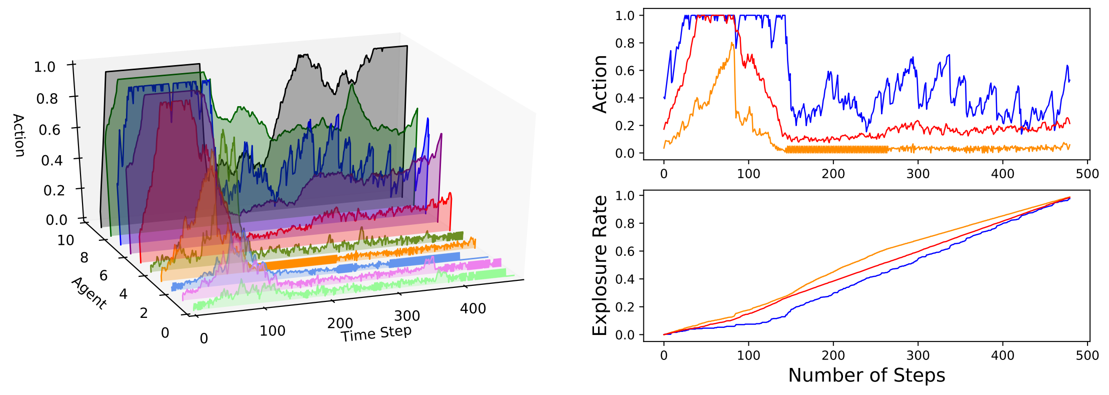


Figure 7: Illustration of 10 randomly selected contracts. (a) Action distribution of each agent. (b) Relation between the actions and exposure rates.

Table 6: Performance comparison on competitive agents.

Method	UR	NR	OR
Random	0.686	0.0	0.231
Static Action	0.884	0.033	0.0
DG	0.329	0.588	0.0
SG	0.823	0.077	0.016
PID	0.402	0.499	0.016
HWM	0.255	0.597	0.065
SHALE	0.598	0.319	0.0
LR	0.357	0.545	0.015
RFR	0.341	0.545	0.031
FeUdal	0.313	0.589	0.015
HMARL	0.179	0.725	0.013

We follow the definition of the smoothness metric used in the previous study [8] to evaluate the smoothness of the compared methods. $\hat{e}_t^n$ denotes the total exposure of contract n until time step t, while $\hat{e}_t^{*n}$ is its optimal smooth delivery amount until time step t, as reflected by the perfect smoothness line. Based on this, the smoothness of a contract is denoted as: $\sigma_n(t) = \frac{e_t^n - e_t^{*n}}{e_t^{*n}}$. We choose the 25-th, 50-th, 75-th, and 95-th percentiles of the sorted $\sigma_n(t)(t \in [0, T])$ to show the smoothness of different methods. The quantitative results of smoothness are shown in Table 5. The main observation is that most agents in HMARL have exposures close to the optimal smooth ones because it is with the smallest σ^{75} and σ^{50} compared to other methods. While for most ads, HMARL achieves better smoothness performance than that of the other methods, it encounters worse values in the 95-th and 25-th percentiles. This phenomenon might be attributed to the fact that HMARL tries to coordinate all the ads, in which some ads with more future impressions will be delayed to display (smaller σ^{25}) and some ads will be allocated to more impressions (larger σ^{95}) so as to make way for some other competition ads with more future impressions.

5 CONCLUSIONS

In this paper, we formulate the problem of large-scale GDAs allocation as a multi-agent reinforcement learning setting. Based on this formulation, we propose the hierarchical multi-agent reinforcement learning framework, which consists of a manager policy and a set of sub-policies. The hierarchies defined over agents enable the sharing of sub-policies across large-scale agents to handle their heterogeneous states. Extensive experiments on three large real-world datasets demonstrate the effectiveness of the proposed framework.

REFERENCES

[1] 2019. eMarketer. https://www.emarketer.com/content/emarketer-total-media-ad-spending-worldwide-will-rise-7-4-in-2018. Accessed: 2010-02-04.

[2] Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. 2014. A dynamic near-optimal algorithm for online linear programming. *Operations Research* (2014), 876–890.

[3] Sanjeevan Ahilan and Peter Dayan. 2019. Feudal Multi-Agent Hierarchies for Cooperative Reinforcement Learning. *arXiv preprint arXiv:1901.08492* (2019).

[4] Karl Johan Åström and Tore Hägglund. 1995. *PID controllers: theory, design, and tuning*. Vol. 2.

[5] Ron Berman. 2018. Beyond the last touch: Attribution in online advertising. *Marketing Science* 37, 5 (2018), 771–792.

[6] Vijay Bharadwaj, Peiji Chen, Wenjing Ma, Chandrashekhar Nagarajan, John Tomlin, Sergei Vassilvitskii, Erik Vee, and Jian Yang. 2012. Shale: an efficient algorithm for allocation of guaranteed display advertising. In *KDD*. 1195–1203.

[7] Zehong Cao and Chin-Teng Lin. 2019. Reinforcement Learning from Hierarchical Critics. *arXiv preprint arXiv:1902.03079* (2019).

[8] Peiji Chen, Wenjing Ma, Srinath Mandalapu, Chandrashekhar Nagarjan, Jayavel Shanmugasundaram, Sergei Vassilvitskii, Erik Vee, Manfai Yu, and Jason Zien. 2012. Ad serving using a compact allocation plan. In *Proceedings of the 13th ACM Conference on Electronic Commerce*. 319–336.

[9] Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. 2017. Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution. In *ICML*. 834–843.

[10] Thomas G Dietterich. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* (2000), 227–303.

[11] Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and S Muthukrishnan. 2009. Online stochastic matching: Beating 1-1/e. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*. 117–126.

[12] Arlington M Fink et al. 1964. Equilibrium in a stochastic n-person game. *Journal of science of the hiroshima university, series ai (mathematics)* 28, 1 (1964), 89–93.

[13] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. 2017. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887* (2017).

[14] Mohammad Ghavamzadeh and Sridhar Mahadevan. 2004. Learning to communicate and act using hierarchical reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*. 1114–1121.

[15] Google. 2011. The arrivals of real-time bidding.

[16] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*. 66–83.

[17] Ali Hojjat, John Turner, Suleyman Cetintas, and Jian Yang. 2014. Delivering guaranteed display ads under reach and frequency requirements. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

[18] Ali Hojjat, John Turner, Suleyman Cetintas, and Jian Yang. 2017. A unified framework for the scheduling of guaranteed targeted display advertising under reach and frequency requirements. *Operations Research* (2017), 289–313.

[19] Nitish Korula, Vahab Mirrokni, and Hamid Nazerzadeh. 2015. Optimizing display advertising markets: Challenges and directions. *IEEE Internet Computing* 20, 1 (2015), 28–35.

[20] Jinna Li, Hamidreza Modares, Tianyou Chai, Frank L Lewis, and Lihua Xie. 2017. Off-policy reinforcement learning for synchronization in multiagent graphical games. *IEEE transactions on neural networks and learning systems* 28, 10 (2017), 2434–2445.

[21] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS*. 6379–6390.

[22] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. 2001. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*. ACM, 246–253.

[23] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. 2012. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review* (2012), 1–31.

[24] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. 2005. Adwords and generalized on-line matching. In *FOCS*. 264–273.

[25] Neville Mehta, Prasad Tadepalli, and A Fern. 2005. Multi-agent shared hierarchy reinforcement learning. In *ICML*.

[26] Vahab S Mirrokni, Shayan Oveis Gharan, and Morteza Zadimoghaddam. 2012. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *SIAM*. 1690–1701.

[27] Igor Mordatch and Pieter Abbeel. 2018. Emergence of grounded compositional language in multi-agent populations. In *AAAI*.

[28] Ronald Parr and Stuart J Russell. 1998. Reinforcement learning with hierarchies of machines. In *NeurIPS*. 1043–1049.

[29] Doina Precup. 2000. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst.

[30] Jason Rhuggenaath, Alp Akcay, Yingqian Zhang, and Uzay Kaymak. 2019. Optimal display-ad allocation with guaranteed contracts and supply side platforms. *Computers Industrial Engineering* 137 (2019), 106071.

[31] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *ICML*.

[32] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* (1999), 181–211.

[33] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *ICML*. 330–337.

[34] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Changjie Fan, and Li Wang. 2018. Hierarchical Deep Multiagent Reinforcement Learning. In *AAAI*.

[35] Erik Vee, Sergei Vassilvitskii, and Jayavel Shanmugasundaram. 2010. Optimal online assignment with forecasts. In *Proceedings of the 11th ACM conference on Electronic commerce*. 109–118.

[36] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2017. FeUdal Networks for Hierarchical Reinforcement Learning. In *ICML*. 3540–3549.

[37] Jun Wang and Shuai Yuan. 2015. Real-time bidding: A new frontier of computational advertising research. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. 415–416.

[38] Chao Yu, Minjie Zhang, Fenghui Ren, and Guozhen Tan. 2015. Emotional multiagent reinforcement learning in spatial social dilemmas. *IEEE transactions on neural networks and learning systems* 26, 12 (2015), 3083–3096.

[39] Weinan Zhang, Shuai Yuan, and Jun Wang. 2014. Optimal real-time bidding for display advertising. In *KDD*. 1077–1086.